UNIVERSITETET I OSLO
Institutt for informatikk

Integrasjon mellom Facebook og VoIP-tjenester

Masteroppgave i Informatikk-studiet med forskningsgruppe:

Distribuerte multimedia systemer (DMMS)

av

Sonam Rani

2. mai 2011

Takk til

Jeg ønsker først og fremst å takke min veileder, Knut Omang, for å ha gitt meg mulighet til å skrive om et så spennende tema. Gjennom oppgaven har Knut gitt meg rom til å utvikle masteroppgaven slik jeg har ønsket, og rådet meg på en veldig god måte. Han har også hjulpet meg med andre utdanningsrelaterte oppgaver og avgjørelser, ved siden av masteroppgaven. Jeg ønsker også å takke Håkon Zahl for hans innspill og uformelle biveileding. Jeg takker også Tarik Cicic og Håkon Bryhni for deres kompetansedeling.

Jeg vil til slutt takke min familie for deres støtte og tro på meg. Jeg takker først min avdøde far Yash Paul, som har vært min inspirasjon for å velge et realfaglig studie. Hans minner gir meg motivasjon til å alltid jobbe hardt. Videre takker jeg min mor Asha Rani, som har gjort det mulig for meg å konsentrere meg kun om min utdannelse. Uten hennes omsorg og kjærlighet ville jeg ikke vært her jeg står i dag. Jeg takker min bror Jimmy Paul, som jeg ser høyt opp til, og som har vært mitt faglige forbilde. Han har vært en viktig akademisk kilde og ressurs i min utdanning.

Avslutningsvis takker jeg kjæresten min, Karan Bhandari. Hans støtte, tålmodighet og forståelse har hjulpet meg å overvinne alle hindringer i min studietid.

Innhold

Figurer

Tabeller

Kapittel 1

Introduksjon

1.1 Motivasjon for oppgaven

VoIP (Voice over IP) har på kort tid blitt et populært alternativ til tradisjonell linjebasert telefoni, siden det både er billigere og tilbyr større fleksibilitet. VoIP-tjenester benyttes i dag av mange brukere, men ikke mange nok for at teknologien kan bli en standard fremfor tradisjonell telefoni. For at en tjeneste skal vokse frem og få en stadig økende brukermasse, trenger tjenesten nettverkseffekt. I denne oppgaven vil jeg se på hva som skal til for å gi VoIP-tjeneser økt positiv *nettverkseffekt*, slik at flere brukere ønsker å ringe med VoIP.

I tillegg til å se på hva som kan gi økt nettverkseffekt til VoIP-tjenester, ønsker jeg å se på hvordan man kan lage en integrasjon mellom VoIP-tjenester og et medium som allerede har en stor brukermasse. Det online sosiale nettverket *Facebook* er et slikt medium som har oppnådd en enorm positiv nettverkseffekt. I oppgaven ønsker jeg derfor også å se på hva Facebook kan gjøre for å gi økt nettverkseffekt til VoIP-tjenester, og hvordan man kan gjøre en integrasjon mellom Facebook og VoIP-tjenester.

1.2 Problemstilling

I denne oppgaven skal jeg vurdere følgende:

> *Hvordan kan integrasjon med Facebook gi økt nettverkseffekt for VoIP-tjenester?*

Det finnes to tilnærminger til en integrasjon mellom Facebook og VoIP-tjenester. Man kan se det som at VoIP integreres i Facebook, slik at tjenesten finnes via Facebook i form av en tredjepartsapplikasjon. På den andre siden så kan det sees som at Facebook integreres i VoIP, slik at tjenesten bruker data fra Facebook. I sammenheng med problemstillingen, vil jeg besvare

underspørsmål som:

Hva må til for å gi økt nettverkseffekt til VoIP-tjenester?

Hvordan kan Facebook gi økt nettverkseffekt til VoIP-tjenester?

Tilbyr Facebook verktøy som tillater integrasjon med Facebook-plattformen, og som kan gi økt nettverkseffekt for VoIP-tjenester?

På hvilken måte kan vi tilby VoIP-tjenester som tillater størst brukermasse, gjennom Facebook-plattformen?

1.3 Mitt bidrag

I oppgaven har jeg sett nærmere på hvordan Facebook kan brukes som en utviklingsplattform. Som en del av dette har jeg implementert et eksempel på hvordan en VoIP-tjeneste kan integreres med Facebook. I implementasjonen har jeg vurdert tilnærminger som tillater flest mulige brukere å benytte tjenesten. Jeg håper at funnene i oppgaven vil inspirere flere forskere innenfor feltet, og bidra til nye ideer og bedre implementasjoner som kan gi økt nettverkseffekt for VoIP-tjenester.

1.4 Oppgavens innhold

Oppgaven er delt inn i tre deler:

- I første del introduserer jeg VoIP og protokoller i VoIP for å forstå adressering og adresseformat i VoIP-tjenester. Videre ser jeg på begrepet nettverkseffekt, og hvilken betydning det har for fremveksten av VoIP. Deretter introduserer jeg Facebook for å se hvordan nettverkseffekt fra dette mediet kan brukes til å styrke VoIP-tjenester, slik at tjenesten har et utgangspunkt for å bli en standard teknologi for telefoni.

- I den andre delen av oppgaven introduserer jeg Facebook som en utviklingsplattform, og tar en nærmere titt på verktøy som tilbys av Facebook for å tillate tredjepartsutviklere å bygge applikasjoner i Facebook, og hva som kan brukes til en VoIP-integrasjon.

- I den siste delen av oppgaven beskriver jeg en prototype av en applikasjon i Facebook som integrerer VoIP-tjenester. Jeg ser på hvordan

integrasjonen kan gjøres på en måte som gir mest mulig nettverks-effekt til VoIP-tjenesten, ved å la prototypen tillate flest mulig brukere å benytte tjenesten. Avslutningsvis ser jeg på funnene som har blitt gjort i oppgaven, personlig erfaringer, og beslektet litteratur.

Kapittel 2

Voice over IP (VoIP)

2.1 Hvorfor VoIP?

VoIP er en metode for å bygge opp telefonsamtaler over et IP-nett, i stedet for det tradisjonelle linjebaserte telefonnettverket. I den siste tiden har det vært en sterk trend i bruk av VoIP-tjenester, og slike tjenester har en stadig fremvekst. Å plassere en VoIP telefonsamtale går først og fremst ut på å sette opp kanaler også omtalt som sesjoner, for signalering [1] og mediaoverføring. Deretter vil de analoge stemme-signalene konverteres til digital form, og eventuelt komprimeres. Etter at dataene er digitalisert, vil de segmenteres inn i IP-pakker, og sendes via kanalene over det pakke-svitsjede nettverket. På den mottakende siden vil samme prosedyre reprodusere den originale stemme-strømmen.

En av de mest lovende fordelene med et VoIP-nettverk i motsetning til tradisjonelle linjebaserte nettverk, som det offentlige svitsjede telefoninettet (Public Switched Telephone Network (PSTN)), er at kostnadene kan reduseres betraktelig både i kommunikasjon og i infrastruktur. I VoIP-nettverk vil overføringsmediet, IP-nettverket, være i stand til å håndtere både data- og voice-pakker. Tradisjonelle linjebaserte nettverk vil på den annen side, bære data- og voice-pakker på fysisk separerte nettverk[77]. Konferansetelefoni, viderekobling, automatisk oppringning, ringer-ID og lignende er funksjonaliter som brukere vanligvis belastes for i PSTN-nettverk. I VoIP-nettverk åpnes det for at slike tjenester kan være kostnadsfrie.

Den største forskjellen mellom VoIP-telefoni og tradisjonell telefoni er måten brukere faktureres på. I linjebaserte nettverk vil brukeren bli belastet for antall minutter han er i en samtale, altså hvor lenge brukeren er koblet til nettverket. I VoIP-tjenester og fakturerte Internett-baserte tjenester generelt, så er det naturlig å belaste brukeren etter hvor mange Megabyte med data som overføres over IP-nettverket. I en praktisk sammenligning er det

[1] Signalering i telekommunikasjon går ut på informasjons-utveksling angående etablering og kontroll av en telekommunikasjons krets, og håndtering av nettverket.

vist at det er betydelig billigere å ringe med VoIP i et gitt tidsintervall, i motsetning til å ringe med tradisjonell telefoni, i løpet av samme tidsperiode.

Ettersom VoIP-tjenester bruker et IP-nettverk som transmisjonsmedium, så kan et VoIP-system interagere med andre tjenester tilgjengelig over IP-nettet, eksempelvis videosamtale, audiokonferanse, og annen type utveksling av informasjon. VoIP-tjenester kan lages lokasjonsuavhengig, som gjør at det eneste brukeren trenger for å koble seg til et VoIP-system, er en rask og stabil Internett-forbindelse. I et VoIP-nettverk så er det også mulig for brukere å sende flere samtaler over bredbåndsforbindelsen, uten å legge til flere linjer, som man ville behøvd i et linjebasert nettverk [77]. Slike VoIP-systemer kalles for *rene VoIP-systemer*. Det finnes imidlertid linjebaserte nettverk som har en mer tilpasset arkitektur og bruker VoIP i kjernenettverket. Slike systemer som tilpasser VoIP i linjebaserte telefonsystemer, kalles for *hybrid VoIP-systemer*. I denne oppgaven omtaler jeg «rene» VoIP-systemer.

VoIP er en *Best effort*[2]-tjeneste, som ikke tilbyr noen garantert *tjenestekvalitet (Quality of Service (QoS))*. Dette gjør at VoIP-implementasjoner kan stå ovenfor utfordringer som *pakkeforsinkelse*, *pakketap* og *Jitter*. VoIP-jitter omtales som tidsvariasjoner i sending og mottakelse av pakker, altså at forsinkelsen i levering av pakkene varierer. Som resultat av disse variasjonene påvirkes kvaliteten av samtalen, slik at den mottakende parten opplever hakk eller «jittering» i samtalen. Pakkeforsinkelse kan årsakes og påvirkes av blant annet den fysiske distansen pakken skal transmitteres over, antall ruting-hopp, krypteringer, voice/data-konvertering, og gjør at mottakeren kan oppleve en pause i samtalen. Pakketap kan inntreffe når store mengder trafikk passerer gjennom nettverket slik at det blir overbelastet eller mettet, og tvinges til å droppe enkelte pakker. Effekten av pakketap er at den tilsvarende delen av samtalen blir droppet, eller at den oppleves som svak.

Selv om VoIP ikke har noen garantert kvalitet på sine tjenester, finnes det imidlertid måter å optimalisere tjenesten på, og tilby brukere tjenestekvalitet med VoIP.

2.2 Protokoller i VoIP

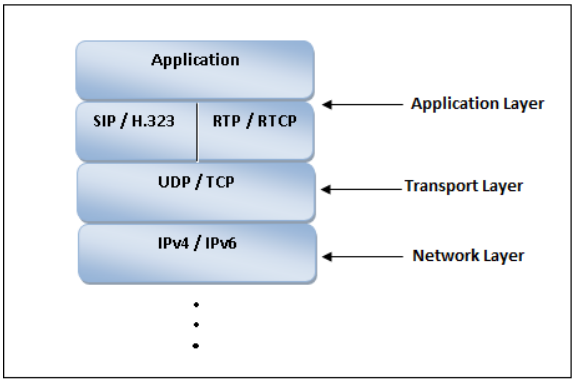
VoIP-kommunikasjon blir regulert av protokoller. Det finnes en rekke protokoller som støtter overføring av multimedia sesjonsdata og sanntidsdata, slik som voicedata som overføres i en VoIP-sesjon. Et eksempel på en slik mediaoverføringsprotokoll, er *Real-Time Transport Protocol (RTP)*. RTP er en applikasjonlagsprotokoll som tilbyr en ende-til-ende transportering av direkteavspillingsmedia (streaming media) over IP-nettverk [19]. RTP brukes

[2]En best effort tjeneste er slik at tjenesten gjør så godt den kan med å levere tjenestekvalitet, men det finnes ingen garanti.

ofte i sammenheng med transportering av audio, video eller simuleringsdata, som i telefoni, videokonferanse og online spill.

RTP foretar ingen ressursreservasjon og garanterer ingen tjenestekvalitet for sanntidstjenester. RTP brukes derfor ofte sammen med protokollen, *Real-Time Transport Control Protocol (RTCP)*. I mens RTP bærer mediastrømmen, brukes RTCP til å monitorere transmisjonen, for å tilby transmisjonsstatistikk og kontrollinformasjon. Slik informasjon kan være antall pakketap, jitter-verdier og forsinkelser om RTP-strømmen, som gjør at VoIP-applikasjonen kan tilpasse flyten bedre. På denne måten vil VoIP-tjenesten optimaliseres, selv om den i utgangspunktet er en tjeneste uten garantert tjenestekvalitet. Både RTP og RTCP er designet til å være uavhengig av underliggende transportprotokoll, og kan derfor kjøres over både *User Datagram Protocol (UDP)* [16], og *Transmission Control Protocol (TCP)* [17] [19].

RTP er et av de tekniske fundamentene i en VoIP-implementasjon, og er i en slik kontekst brukt med en *signaleringsprotokoll*. Ettersom at brukere man vil dele en VoIP-sesjonen med kan befinne seg på forskjellige endepunkter i nettverket, trenger vi en måte å «oppdage» disse endepunktene. Signaleringsprotokoller tillater Internett-endepunkter å oppdage andre endepunkter og å bli enige om en karakterisering av sesjonen de ønsker å ta del i. Det finnes to dominerende signaleringsprotokoller som brukes i VoIP, *H.323* og *Session Initiation Protocol (SIP)*. SIP har blitt standardisert av *Internet Engineering Task Force (IETF)* for invitasjon til multicast konferansesamtaler, multimedia-distribusjon og Internett-telefoni [18]. Et eksempel på en VoIP-stakk kan se slik ut:



Figur 2.1: Et eksempel på en VoIP-stakk.

I denne oppgaven vil jeg fokusere på SIP som signaleringsprotokoll. For mer om H.323, se [27].

SIP-protokollen er en applikasjonsnivås signaleringsprotokoll brukt til

å etablere, modifisere og terminere både unicast[3] og multicast[4] multimediasesjoner. SIP kan også invitere deltakere til allerede eksisterende sesjoner. Følgende funksjonalitet støttes av SIP for etablering og terminering av multimedia kommunikasjon [18]:

Brukerlokalisering - determinering av endesystemet som skal benyttes for kommunikasjon.

Brukertilgjengelighet - determinering av viljen til mottaker-parten for å engasjere i kommunikasjon.

Brukerkapasitet - determinering av media og media parametre som skal benyttes.

Sesjonsoppsett - etablering av sesjons-parametre ved både oppringer og mottaker.

Sesjonshåndtering - inkludert overføring og terminering av sesjoner, modifisering av sesjonsparametre, og forespørsel om tjenester.

E.164[66] er en anbefalning fra *International Telecommunication Union* som definerer det standarde internasjonale nummereringsplanet for telekommunikasjon, og som blant annet brukes i PSTN-nettverk. Standarden definerer blant annet antall sifre, og hvordan formatet skal være på telefonnumre i henhold til standarden. De fleste VoIP-implementasjoner støtter E.164, og tillater VoIP-brukere å ringe til og fra PSTN-nettverk. Brukere kan i en slik implementasjon dermed ringe til fasttelefonnumre og mobiltelefonnumre fra VoIP systemet.

Noen implementasjoner av VoIP tillater derimot andre teknikker for å identifisere brukere. Ettersom det er signalerinsprotokoller som skal håndtere lokaliseringen av andre brukere som man vil dele en VoIP-sesjon med, må en slik identifikasjon være på et format som signaleringsprotokollene forstår. VoIP-implementasjoner som bruker SIP som signaleringsprotokoll adresserer brukerne med såkalte *SIP URIer*. Til forskjell fra slike URIer, så inneholder ikke telefonnumre i henhold til E.164-standarden noe informasjon om hvor forespørsler skal rutes. I tilfeller hvor brukere ringer til numre i henhold til E.164-standarden, så vil det gjøres en oversettelse fra E.164-identifikatoren til en ikke-E.164 identifikator, ved hjelp av standarder som *ENUM (E.164 Number Mapping)*. Eksempelvis kan en VoIP-bruker slå nummeret «+1276032600», og mappingen vil gi URI'en «sip:user@carrier.com», som gir hint om hvilken retning informasjon skal sendes [21].

I neste seksjon skal vi se mer på SIP og signalering med SIP-URIer.

2.2.1 Mer om SIP og SIP URIer

Et SIP-system består som regel av to type komponenter: *Brukeragenter (UA)* og *Nettverksservere*. En brukeragent representerer et endesystem som handler på vegne av en bruker som ønsker å delta i sesjoner med SIP-signalering.

[3] En Unicast sesjon er en sesjon mellom en avsender og en mottaker.

[4] En Multicast sesjon er en sesjon mellom en avsender og flere mottakere, eller flere avsendere og flere mottakere.

En SIP brukeragent har to fundamentale funksjoner: Å lytte etter innkommende SIP-beskjeder, og å sende SIP-beskjeder ved brukerhandlinger eller til innkommende beskjeder. Når brukeragenten sender en SIP-forespørsel, tar den rollen som en *brukeragent-klient (UAC)*. Når brukeragenten mottar en SIP-forespørsel og genererer en SIP-respons, inntar den rollen som en *brukeragent-server (UAS)*. Rollene som UAC og UAS som brukeragenten inntar, varer så lenge SIP-sesjonen eksisterer [18].

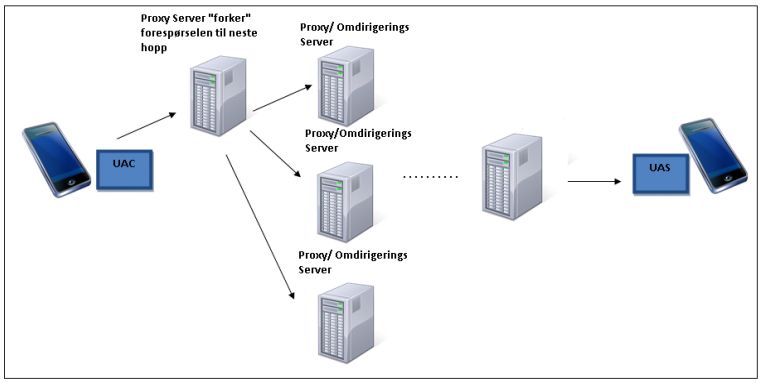
Hovedfunksjonen til SIP nettverksservere er å tilby *navnoppslag* og *brukerlokalisering* [22]. Når en oppringer ønsker å plassere et anrop, vil brukeragenten på oppringerens SIP-system i endepunktet sende en SIP INVITE-forespørsel til brukeragenten mot mottakerens SIP-system, for å invitere til en sesjon. Endepunkt-adressene er logisk eller lokale adresser, da brukere kan befinne seg på forskjellige lokasjoner, og koble seg til fra forskjellige enheter. I SIP brukes en e-post liknende *global* identifikator, kalt SIP URIer eller *SIP-adresser* for å identifisere en bruker uavhengig av lokasjon. En bruker kan dermed ha flere *lokale* kontaktadresser som sier noe om brukerens nåværende endepunkt, og som er assosiert til brukerens globale SIP-URI. Eksempelvis kan en SIP-URI være *sip:kari.nordmann@domene.com*, som globalt skal identifisere brukeren Kari Nordmann. Den mappede lokale kontaktadressen til Kari fra verten eller IP-adressen til endesystemet hun på nåværende tidspunkt er logget på, kan være *sip:kari.nordmann@vertsnavn* eller *sip:kari.nordmann@ipadresse*. På et annet tidspunkt kan brukeren logge på via en annen maskin på et annet endepunkt, og den lokale kontaktadressen vil da være en annen.

Generelt vil oppringeren ikke vite IP-adressen eller vertsnavnet til brukeragenten på mottakerens endesystem, oppringeren vil typisk kun ha kjennskap til SIP-URIen som representerer mottakeren. Ved å benytte denne identifikatoren kan brukeragenten til oppringeren spørre SIP nettverksservere om hvem som er i stand til å oppløse identifikatoren til en lokal kontaktadresse til endesystemet til mottakeren. Prosessen av å finne den korresponderende IP-adressen eller verten til en SIP-URI, kalles navnoppslag [22, 18].

SIP har to måter å sende trafikk på, som gir en *logisk* adskilling i to type nettverks-servere: *SIP Proxy-servere* og *SIP Omdirigerings-servere* [18].

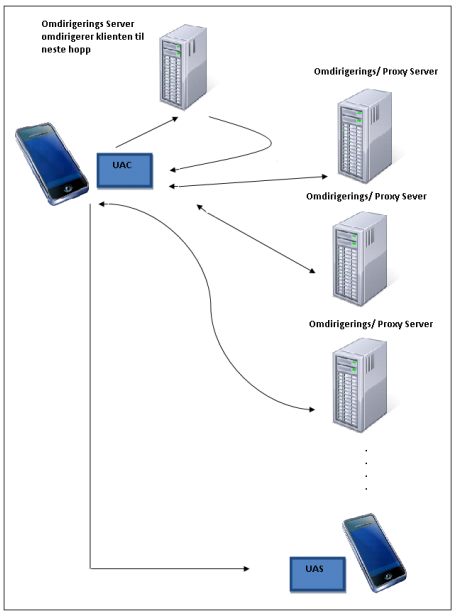
En proxy-server vil typisk motta en SIP forespørsel, avgjøre hvilken SIP-server som kan nå mottakeren av forespørselen, og så videresende forespørselen til den SIP-serveren. Når forespørselen til slutt når mottakerens brukeragent på endesystemet, vil brukeragenten generere en respons på forespørselen, og den vil sendes tilbake langs ruten den kom på, til brukeragenten på avsenderens endesystem. Som et resultat av slike avgjørelser om ruting til «neste hopp», så kan hende at en server avgjør at flere SIP-servere er i stand til å nå mottakeren. I slike tilfeller så tillater SIP å *forke* den innkommende forespørselen. Dette gjør at forespørselen videresendes i parallell til flere servere [22, 18]. En proxy-server har ingen måte

å vite om neste server er en proxy-server, en omdirigerings-server eller en brukeragent på et endesystem. Med denne metoden kan SIP-forespørseler derfor traversere mange servere på veien fra brukeragenten på oppringerens endesystem til brukeragenten på mottakerens endesystem, noe som ikke er skalerbart. På den annen side så vil det være mulig for en bruker å finne en annen bruker ved kun å kjenne til SIP-URIen i stedet for å vite IP-adressen eller verten til maskinen til brukeren som han ønsker å ringe. Proxy-serverne fungerer som et bindeledd mellom brukeragentene til to kommuniserende parter, hvilket betyr at en SIP proxy-server også kan brukes til å skjule lokasjonen til en bruker [22].



Figur 2.2: SIP Proxy-server funkjsonalitet.

En omdiringerings-server mottar SIP-forespørseler, men i stedet for å videresende forespørselen, så forteller den brukeragenten til oppringerens endesystem hvilken SIP-server som er kan nå mottakeren. Omdirigeringsservere sender rutinginformasjon tilbake til brukerklienten som forespørselen kom fra, og får brukerklienten til å kontakte neste server direkte. På denne måten tar SIP-serverne seg selv ut av transaksjonen, men hjelper samtidig med å lokalisere mottakerens endesystem. Omdirigeringsserverne svarer med en omdirigeringsrespons på forespørselen fra brukerklienten, som inneholder adressen til serveren til neste hopp, etter å ha avgjort hvem server som kan nå mottakerens endesystem. Ettersom at omdirigerings-servere ikke deltar i hele signaleringstransaksjonen som proxy-servere, så er denne måten å sende trafikk mer skalerbar.



Figur 2.3: SIP Omdirigerings-server funkjsonalitet.

Ettersom forskjellen mellom en proxy-server og en omdirgerings-server er logisk, kan en SIP-server ha begge konfigurasjonene implementert og veksle mellom trafikksendings-metodene [18]. Brukeragenten på oppringerens endesystem kan dermed sende forespørselen til en SIP nettverksserver som enten videresender forespørselen (proxy-server) eller omdirigerer brukeragenten (omdirigerings-server) til ytterligere SIP-servere nærmere mottakerens endesystem, helt til den kommer frem til en SIP-server som definitivt vet vertsnavnet eller IP-adressen til maskinen som mottakeren sitter på, og deretter starte sesjonsoppsettet. For mer om SIP, se [22, 18].

VoIP-tjenester kan tilbys via SIP-systemer. Slike SIP-systemer finnes i form av *SIP-telefoner* eller *SIP-softtelefoner (softphones)*, også omtalt som *SIP-klienter*. En SIP-telefon er en maskinvarebasert SIP-brukeragent som ligner på en tradisjonell telefon, og tilbyr de tradisjonelle anropsfunksjonalitene til en telefon, som å taste nummer, svare/avslå anrop, parkere/hente anrop, viderekoble anrop og lignende. En SIP-klient er en programvarebasert SIP brukerklient som kan installeres på en PC/datamaskin eller en mobil enhet, som laptoper, PDAer og mobiltelefoner. Tale og lytting i softtelefoner støttes ved bruk av headset med mikrofon og/eller et lydkort, dersom høytalere og mikrofon ikke er innebygd i enheten. Brukeren blir ofte presentert med et grensesnitt som tillater brukere å svare/avslå anrop og lignende anropsfunksjonaliteter. I dag finnes det SIP-programvare for Linux, Mac, Windows og mobile (smarttelefon) miljøer, det finnes også

SIP-programvare som fungerer på tvers av plattformer [75, 18].

Selve signaleringsprosessen, sesjonsoppsettet og mediaoverføringen håndteres som regel av programvaren. Dette gjør at brukere av en slik VoIP-tjeneste kun trenger å ha kunnskap om den globale SIP-URIen som identifiserer brukeren han ønsker å ringe, for å plassere et anrop. I oppgaven fokuserer jeg på VoIP-tjenester i SIP-klienter, og SIP-URIer som brukeridentifikasjon.

Kapittel 3

Nettverkseffekt

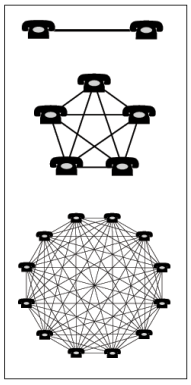
3.1 Om nettverkseffekt

Nettverkseffekt omtales som effekten en bruker av et produkt[1] har på *verdien* av produktet, for andre brukere og andre mennesker [74]. Når et produkt oppnår nettverkseffekt, kalles konsekvensen av nettverkseffekten for *nettverkseksternalitet*. Denne eksternaliteten kan være en *positiv eksternalitet* eller *negativ eksternalitet*. Positiv eksternalitet er dersom konsekvensen av nettverkseffekten er slik at verdien til produktet øker. I motsetning så er negativ eksternalitet slik at konsekvensen av nettverkseffekten gjør at produktet minker i verdi, og omtales også som *metning*. Som en følge av positiv eksternalitet til en konkurrent i et marked, kan det forekomme en korresponderende negativ eksternalitet til alle andre konkurrenter i markedet [74].

Et klassisk eksempel på å demonstrere nettverkseffekt og nettverkseksternalitet, er telefoni. Når telefoni er slik at fordi folk bruker telefon, så fører det til at andre ønsker å skaffe seg en telefon, så har telefoni oppnådd en nettverkseffekt. Altså, andre brukere kjøper seg telefoner ikke for å øke verdien til telefonen for andre eiere av telefoner, men gjør det allikevel. Positiv eksternalitet er at ettersom flere kjøper seg telefoner, så vil telefonen til hver enkelt bruker få økt verdi for eieren. Et eksempel på positiv eksternalitet er at som resultat av at flere mennesker kjøper seg telefoner, så kan eiere av telefoner oppleve at man enklere kan komme i kontakt med andre og tilgjengeliggjøre seg, og derfor bærer telefonen på seg til enhver tid. Et annet eksempel er at eiere stadig kjøper seg nye og bedre telefoner, fordi telefonen som et produkt er mer verdifull for eieren. En negativ eksternalitet ville vært at som konsekvens av at telefoni har oppnådd nettverkseffekt slik at flere brukere kjøper seg en telefon, så vil telefonen minke i verdi for eksisterende eiere av telefoner. Eksempel på negativ eksternalitet er at som resultat av at flere mennesker har kjøpt seg telefon, så kan telefonei-

[1] Et produkt refererer i denne sammenhengen til en gode eller tjeneste.

ere oppleve at de tilgjengeliggjør seg altfor mye, som gjør at brukere velger å gå mindre med telefonen på seg. Et annet eksempel er at brukeren ikke ønsker å oppgradere eller vedlikeholde telefonen, fordi telefonen som et produkt er mindre verdifull for eieren. Begrepet, nettverkseffekt, anvendes som regel i sammenheng med positiv eksternalitet [74, 46].



Figur 3.1: Figur fra Wikipedia: Illustrasjon av positiv nettverkseffekt.

Positiv nettverkseffekt kan utløse en *kjedereaksjon* eller *bølgeeffekt* i den forstand at flere mennesker gjør eller tror på en ting, på grunnlag av at mange andre mennesker gjør det samme. Denne bølgeeffekten kan også gi opphav til en *positiv løkkeeffekt*, som er slik at A produserer mer av B, som igjen produserer mer av A. En positiv løkkeeffekt i eksempelet over ville vært at telefoni skaper flere telefoneiere, som igjen skaper nye og bedre telefoner.

3.2 VoIP og nettverkseffekt

VoIP er en moderne form for telefoni, og som med tradisjonell telefoni som ble introdusert på 1800-tallet, så behøves positiv nettverkseffekt for at teknologien kan vokse ytterligere frem, og bli en standard for telekommunikasjon. I denne seksjonen skal vi se på hva som må til for at VoIP-tjenester skal få en økt nettverkseffekt.

Brukere er som regel opptatt av at ting skal være «enkelt», slik at hvis brukere skal benytte et produkt eller en tjeneste, trenger det å oppleves som enkelt for brukeren å benytte nettopp dette produktet eller denne tjenesten. Når man skal «overtale» brukere til å benytte et nytt produkt eller ny tjeneste, er det derfor spesielt viktig at brukeren opplever tjenesten som lettvin å bruke.

Signaleringsprotokoller gir mulighet for mer fleksible brukeradresser. Som vi har sett, så kan VoIP-implementasjoner ha et annet adresseringsformat enn et telefonnummer som i tradisjonell telefoni og andre formater

i henhold til E.164-standarden. I en ren VoIP-tjeneste som bruker SIP som signaleringsprotokoll, vil brukere identifiseres med en global SIP URI. SIP URIer i VoIP med SIP skal identifisere brukere, tilsvarende som telefonnumre identifiserer brukere i tradisjonell telefoni. Slike SIP URIer kan være på formen:

```
sip:navn@domene.com
sip:nummer@domene.com
```

I eksempelet over så er *domene.com* domenet til brukerens tjenesteleverandør (ITSP).

Som en følge av at VoIP-implementasjoner kan bruke egne adresseformat som er forskjellig fra brukeres tradisjonelle telefonnumre, så skaper dette noen utfordringer for hvordan brukere av slike VoIP-tjenester kan kontakte andre brukere. Selv om det finnes en stor masse med brukere som eier SIP-adresser og et godt utvalg av SIP-telefoner og SIP-klienter, så er det vanskelig å få kunnskap om andres SIP-adresser, da det ikke finnes noen stort og offentlig oppslagsverk som opprettholder og tillater brukere å slå opp SIP-adresser. I motsetning til tradisjonell telefoni, har ikke VoIP noen form for telefonkatalog, telefonregister eller opplysningstjeneste som tillater brukere å finne andre brukeres SIP-adresser på en «enkel» måte. Adressebøker internt i SIP-klienter gjør at brukere av samme SIP-klient eller SIP-telefon kan holde på hverandres SIP-adresser, i mindre mengder. Mange SIP-klienter tillater også brukere å importere kontakter fra andre eksisterende adressebøker, som Microsoft Outlook. Imidlertid behøves det et større og felles kontaktoppslagsverk, som tillater større masser med brukere å oppgi sine SIP-adresser, og å hente ut andres SIP-adresser.

Altså, en av optimaliseringene som kan gjøres for slike VoIP-tjenester, er å lage et stort og felles kontaktoppslagsverk hvor brukere kan oppgi og slå opp andres SIP-adresser. Dette vil gjøre det enklere for brukere å få kunnskap om kontaktinformasjon til personer de ønsker å ringe, og kan gjøre at flere brukere vil benytte VoIP.

Så la oss videre anta at det finnes et slikt stort og felles kontaktoppslagsverk, og at oppslagsverket finnes online. Brukere kan da eventuelt logge inn i kontaktoppslagstjenesten, og legge inn sin kontaktinformasjon. Når brukeren ønsker å plassere et anrop til en gitt person, finner han personens kontaktadresse, gitt at personen har oppgitt den til oppslagstjenesten. Videre så kan brukeren starte et anrop til personen. Prosessen videre er avhengig av om brukeren benytter en SIP-klient eller en SIP-telefon. Dersom brukeren benytter en SIP-telefon, vil han taste inn kontaktadressen han fant i oppslagsverket på Internett, tegn for tegn over i SIP-telefonen sin, eventuelt lagre adressen i telefonens adressebok, og starte anropet. Dersom brukeren benytter en SIP-klient så slipper brukeren å taste kontaktadressen tegn for tegn inn i klientens ringegrensesnitt, brukeren kan kopiere den og lime den inn i ringefeltet, eventuelt lagre adressen i klientens adressebok, og starte anropet.

Selv om prosessen av å gjøre adresseoppslag, bringe adressen til telefonen eller klienten, og så starte anropet, ikke virker så veldig tungvint, så er det imidlertid ikke en enklere måte å ringe på enn med tradisjonell telefoni. I starten vil det være naturlig for brukere å gjøre mange slike adresseoppslag for å fylle adressebøkene med kontaktadressene til de fleste andre personer som brukeren ønsker å ringe med VoIP. For å forenkle prosessen av å plassere en samtale med VoIP, så bør kontaktoppslagsverket også tillate brukere å ringe direkte fra oppslagstjenesten. Dette vil gjøre at tjenesten gjør adresseoppslag til en gitt person, og lar brukere initiere en samtale i to enkle steg.

For å få flere brukere til å benytte VoIP-tjenester og potensielt gi tjenesten en økt nettverkseffekt, så må vi altså gjøre VoIP enkelt for brukere. Ettersom det finnes mange brukere med SIP-adresser og nok leverandører av SIP-klienter, så må vi tillate brukere å finne hverandres kontaktinformasjon på en enkel måte. Dette kan gjøres ved å lage et stort og felles kontaktoppslagsverk. Videre så må VoIP-tjenester også tilgjengeliggjøres mer for brukere, slik at de like enkelt kan ringe med VoIP som med mobiltelefonen. Dette kan oppnås ved å integrere kontaktoppslagstjenesten med en eller flere SIP-klienter, som tillater brukere å ringe andre brukere direkte fra oppslagstjenesten. Det behøves en slags «slå opp-klikk-og ring»-mekanisme for VoIP, slik som i tradisjonell telefoni.

Vi skal videre se på noen arenaer for å bygge en slik kontaktoppslagstjeneste som slår opp brukeres SIP-adresser for hverandre, og tilgjengeliggjør VoIP-tjenester.

3.3 Online sosiale nettverk

Et *online sosialt nettverk* er en webside, tjeneste eller plattform på nett, som fokuserer på å la mennesker uttrykke deres individualitet, og å bygge og reflektere sosiale nettverk eller sosiale relasjoner mellom mennesker [76]. De fleste sosiale nettverkstjenester er webside-baserte, og kalles for *sosiale sider* eller *sosiale nettsteder*. Et online sosialt nettverk regnes som et storskalert system som er i kontinuerlig utvikling [2].

Sosiale nettsteder bruker å ha noen konvensjonelle funksjoner til felles. Et typisk sosialt nettsted består essensielt av representasjoner av hver bruker, hvor en brukerrepresentasjon ofte er i form av en *brukerprofil*. En slik profil kan inneholde forskjellig personlig informasjon om brukeren. Dette kan være informasjon som kontaktinformasjon, interesser, utdanning, arbeids, sivilstatus, religion og lignende. Brukere kan som regel laste opp bilder av seg selv til deres profiler, og publisere annet innhold, som video, audio, og blogger. Typisk består brukerprofiler av en egen seksjon som tillater andre brukere å kommentere og publisere innhold til brukerens profil. De fleste sosiale sider tilbyr også brukere mulighet til å interagere på et mer

personlig plan, som via e-poster og direktemeldinger. For å beskytte brukeres personvern, så tilbys som regel kontroller som tillater brukere å styre hvem som har innsyn i deres brukerprofiler, hvem som kan kontakte dem og lignende. Utover disse konvensjonelle funksjonene, har de fleste sosiale nettsteder ytterligere tilleggstjenester individuelt for hvert nettsted [76].

Det sosiale nettverket bygges typisk opp ved å la brukere bygge relasjoner og koble sammen brukerprofilene deres, med hverandre. Slike sosiale linker kan opprettes mellom blant annet venner, kolleger, familie, folk som deler felles interesser, nasjonalitet og religion. Sosiale nettverk gjør altså at brukere kan bygge relasjoner på forskjellige nivåer: både på et familienivå, og også på et nasjonalt eller internasjonalt nivå.

Studier viser at de fleste PC-brukere og/eller Internett-brukere har enten tatt del eller er en del av ett eller flere nettbaserte sosiale nettverk [2]. Sosiale nettsteder viser seg å være spesielt populært blant yngre Internettbrukere. Sosiale nettverk er derimot også blitt populært innen forretning, da mange selskaper bruker slike sosiale sider til å markedsføre og promotere deres produkter. Noen bedrifter bruker til og med de sosiale nettstedene som distribusjonskanaler. I dag regnes *Facebook*, *MySpace*, *Twitter* og det profesjonelle sosiale nettverket, *LinkedIn*, som de største sosiale sidene i verden, da Facebook regnes som det største av dem [2, 76].

3.4 Facebook og nettverkseffekt

I likhet med telefoni, må positiv nettverkseffekt til for at flere mennesker skal tilslutte seg et sosialt nettverk, og for at brukere av det sosiale nettverket skal benytte deres brukerprofiler aktivt. Verdien av det sosiale nettverkets brukerprofil for hver av nettverkets brukere, øker ettersom antall brukere av det sosiale nettverket, øker. Altså, jo flere som oppretter brukerkontoer til nettstedet, dess mer verdi får brukerkontoen for brukere av nettstedet, da flere mennesker kan nås og flere relasjoner kan bygges [15].

Facebook regnes som å være dagens største online sosiale nettverket [76, 15]. Verdien av Facebook for hver bruker øker ettersom antall forbrukere som bruker Facebook, øker. Facebook er en plattform som ikke bare tiltrekker forbrukere, men også *annonsører* og *applikasjonsutviklere*. Facebook tillater annonsører å opprette og tilpasse annonser i Facebook. Annonsører kan rette annonsene mot riktige målgrupper, for en best mulig markedsføring. Facebook tillater også tredjepartsutviklere å utvikle applikasjoner i Facebook, og er de første som åpnet for tredjepartsutvikling mot egen plattform [50]. Applikasjonene kan også på samme måte rettes mot brukergrupper, og inkluderes i Facebook sitt applikasjonsbibliotek, og tilgjengeliggjøres for alle Facebook-brukere.

Ettersom Facebook er det største online sosiale nettverket og bringer sammen tre forskjellige typer brukere, har nettstedet ikke bare sterk nett-

verkseffekt mellom forbrukere (*enkel-sidet* nettverkseffekt), men det har også *multi-sidet* nettverkseffekt. For eksempel, ettersom antall forbrukere som bruker Facebook øker, så dannes det mer verdi for applikasjonsutviklerne siden de har en stadig økende brukermasse. Applikasjonsutviklere ønsker å utvikle applikasjoner i en plattform som har en stor brukerbase. Imidlertid så er det motsatte også sant. Forbrukere foretrekker også å bruke en plattform med flere høykvalitets applikasjoner tilgjengelig, og dermed skapes det en positiv løkkeeffekt. Multi-sidet nettverkseffekt kan sees på som et spesialtilfelle av et *to-sidet marked*[2].

Som følge av slike ekstremt sterke multi-side nettverkseffekter i markedet for online sosiale nettverk, vil det resultere i at en enkel standard vil oppnå dominans i markedet, over tid. På grunn av de positive effektene på den fremvoksende vinneren, vil det oppstå komplementære negative effekter på konkurrentene [15]. Tilsynelatende er det Facebook, med sine 600 millioner aktive globale brukere, som er den fremvoksende dominanten innenfor markedet for nettbaserte sosiale nettverk. Dette betyr imidlertid ikke at de konkurrerende tjenestene vil legges ned. Andre tjenester vil som regel fortsette å vedvare, fordi de leverer et annet såkalt *value proposition* (unikt tilbud), og i noen tilfeller, til en annen målgruppe av forbrukere [15]. Ikke bare finnes det sterk mulit-sidet nettverkseffekt i det online sosiale nettverket, Facebook, i tillegg til nettverkseffekten er det signifikante kostnader forbundet med å bytte til en konkurrerende tjeneste. La oss eksempelvis kun vurdere forbrukergruppen i Facebook. Dersom en bruker velger å gå over til en konkurrerende tjeneste, må brukeren ikke bare forlate Facebook og opprette en konto et annet sted, men brukeren må også overtale en mengde av sine venner til å bytte tjeneste. Sjansene for at det skal skje i en meningsfull skala er såpass liten, at Facebook faktisk tillater sine brukere å eksportere brukerdata til andre tjenester [15].

Så, hvordan kan Facebook gi økt nettverkseffekt til VoIP-tjenester? Facebook som et online sosialt nettverk er en arena hvor det er naturlig å oppgi informasjon som SIP-adresser, ettersom brukerprofilene inneholder lignende kontaktinformasjon og personlig informasjon om brukere av nettstedet. I tillegg til å være en passende arena, så tilbyr Facebook utviklingsmuligheter for tredjeparter til å bygge applikasjoner og tilgjengeliggjøre tjenester som VoIP. Facebook har oppnådd en ekstremt sterk multi-side nettverkseffekt som kan utnyttes av tredjepartsutviklere til å gi nettverkseffekt til tjenestene de integrerer i Facebook-plattformen, ved å introdusere en såpass stor brukermasse. I de neste kapitlene skal vi se på Facebook som et online sosialt nettverk, og hvilke verktøy som tilbys av Facebook for å tillate tredjeparter å integrere med Facebook-plattformen.

[2]Et to sidet marked er økonomiske plattformer som har to distinkte brukergrupper som tilbyr hverandre nettverksfordeler.

Kapittel 4

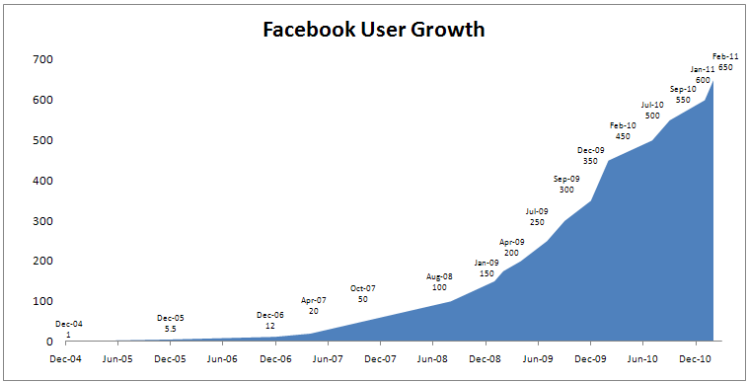
Facebook: et online sosialt nettverk.

Figur 4.1: Facebook-logo.

Facebook er et av flere sosiale medier som er med på å gjøre Internett mer sosialt. Det interaktive nettstedet ble grunnlagt i 2004, og har i dag over 600 millioner aktive[1] brukere, spredt over 180 land med mer enn 70 oversettelser av nettsiden, ettersom tjenesten er flerspråklig. Av disse er 70 % av brukerne utenfor USA og 1,5 millioner er norske brukere. Over 250 millioner aktive brukere aksesserer Facebook via mobile enheter [51, 5, 65].

Facebook-gründeren, Mark Zuckerberg, hevder at deres overordnede mål er *«å gi folket mulighet og makt til å dele informasjon, og gjøre verden mer åpen og forbundet»* [51]. Nettstedet skal hjelpe brukere å kommunisere mer effektivt med venner, familie og kolleger, og gi mulighet til å bygge et større sosialt nettverk. I gjennomsnitt har hver Facebook-bruker 130 vennerelasjoner assosiert med sin brukerkonto, og brukerne benytter sammenlagt over 700 milliarder minutter i måneden på nettstedet [51]. I tabellen nedenfor illustreres brukerøkningen i Facebook fra da nettstedet ble grunnlagt i 2004, til i år. Figuren er tatt fra [10]:

[1] En aktiv bruker regnes som en som besøker nettsiden minst hver 30. dag.



Figur 4.2: Facebook Nettverkseffekt: Økning i antall Facebook-brukere fra 2004-2011.

4.1 Historikk

Navnet til nettsiden stammer fra hverdagsutrykket som ble brukt på bøker som studenter ved Harvard University ble tildelt av universitets administratorer ved semesterstart, for å hjelpe studentene til å bli bedre kjent med hverandre. Facebook ble grunnlagt av Mark Zuckerberg sammen med hans romkamerater og informatikk-medstudenter, Eduardo Saverin, Dustin Moskovitz og Chris Hughes [5]. Websidens målgruppe var opprinnelig begrenset til kun Harvard-studenter, men ble senere utvidet til å benyttes av andre universiteter i Boston-området, som Ivy League og Stanford University. Etter hvert ble Facebook åpent for de fleste universiteter i USA, og det tok ikke lang tid før grunnskoleelever også kunne bruke Facebook-tjenesten.

I dag kan mennesker i en alder over 13år og som har en gyldig e-post adresse, bli en Facebook-bruker. Facebook-tjenestens målgruppe er rettet mer mot ungdom enn mot voksne [31, 52]. Imidlertid brukes nettstedet aktivt av både unge og vokse, og også utallige bedrifter for å promotere og markedsføre sin geskjeft. Mange av de største internasjonale selskapene har i dag en viss form for integrasjon mot Facebook.

4.2 Facebook brukerkonto

En Facebook brukerkonto består av et navn, en gyldig e-post adresse, og et passord. Valgfritt kan man lenke brukerkontoen til flere e-post adresser, og man kan opprette et Facebook-brukernavn. Brukerkontoen kan også linkes til en bestemt faktureringsmåte med tilsvarende faktureringsinformasjon. Brukeres *grunnleggende* eller *basis* informasjon, omfatter profilbilde,

Facebook-ID, navn og brukernavn. Brukere kan utbrodere profilene sine med mer personlig informasjon eller *utvidet* informasjon, som blant annet fødselsdag, borgerskap, utdanning, arbeid, politisk standpunkt, religiøs tilhørighet, kontaktinformasjon, bilder og videoer.

Alle Facebook-brukere kan sende en *venneforespørsel* til andre Facebook-brukere for å opprette en sosial link mellom brukerne, eller «bli venner» på Facebook. Brukere kan også personliggjøre profilen sin ved å opprette eller melde seg inn i interessegrupper, fansider og hendelsessider. Slike sider er representert med egne profiler eller sider (pages) i Facebook. Interesse- og fan-sider kan være om alle mulige produkter og fenomener. Hendelsessider tillater brukere å lage invitasjoner og formidle informasjon om forskjellige begivenheter som skal skje. Noen av sidene er opprettet av Facebook-utviklere eller Facebook-brukere, mens andre kan være opprettet av organisasjoner eller bedrifter for annonsering og markedsføring.

Figur 4.3: Facebook Brukerprofil.

Facebook brukerprofiler fremviser brukerens profilbilde, vennerelasjoner, grunnleggende og eventuell utvidet informasjon, i tillegg til en *wall*. Brukerens vegg benyttes av brukeren selv og brukerens venner, til å publisere innhold. I tillegg til brukerprofilen, er hver bruker utstyrt med en *hjemside*. Denne siden fungerer som en samleside for brukerne. I hjemsiden vises en *nyhetsstrøm*. Nyhetsstrømmen endrer seg relativt ofte, da den inneholder høydepunkter i den siste tiden. Eksempler på informasjon som dukker opp i nyhetsstrømmen, er blant annet profilendringer, kommende hendelser og bursdager, kommentarer, innlasting av bilder og videoer, innholdspubliseringer og annen informasjon om brukerens venner [44].

4.2.1 Kommunikasjon

Kommunikasjon mellom venner og andre brukere kan gjøres via private eller offentlige beskjeder, eller ved bruk av sanntidsbeskjeder. Brukerkontoene er utstyrt med interne beskjedbokser, slik at en bruker kan sende og motta beskjeder privat. Disse beskjedboksene er konstruert typisk som tradisjonelle e-post kontoer, hvor man har en innboks, en utboks, og kan sende en beskjed til en eller flere mottakere av gangen.

En offentlig beskjed sendes ved å benytte brukerens wall. Når en bruker besøker en annen brukers brukerprofil kan han publisere et innlegg på veggen, og se alle andre vegginnlegg som er blitt publisert på veggen av andre brukere. En bruker kan også selv publisere innhold på veggen sin. Selvpublisert innhold på brukerens egen vegg kalles for *statusoppdateringer*. Statusoppdateringene tillater brukere å dele personlig innhold med andre brukere, og inneholder som regel informasjon om brukerens hverdagslige aktiviteter, som hva brukeren gjør for øyeblikket eller hva han skal i nærmeste fremtid. Både veggpubliseringer og statusoppdateringer vil vises i nyhetsstrømmen til brukerens venner.

Facebooks chat-funksjon er veldig lik tradisjonelle chat-klienter, hvor brukere får opp en liste over andre brukere som er pålogget for øyeblikket. Brukere kan starte en privat sanntidssamtale med en eller flere av de påloggede brukerne.

4.2.2 Andre applikasjoner

Brukerprofilenes vegger, meldingsbokser, interesse- og fan-sider, chat, bilde og video-opplasting er eksempler på *applikasjoner* som er lagt til brukerprofilene, og som er laget av Facebook-utviklere. Facebook består i hovedsak av applikasjoner som ikke er utviklet av Facebook-utviklere, men av tredjepartsutviklere, og som brukere kan legge til i tillegg til de applikasjonene som kommer med som en standard til brukerprofilene. Slike applikasjoner kan være gratis-applikasjoner eller fakturerte applikasjoner, avhengig av utvikleren og applikasjonens funksjonalitet. De standarde applikasjonene som følger med brukerprofilene, og de fleste applikasjoner utviklet av Facebook-utviklere generelt, er kostnadsfrie. Tredjepartsutviklere av applikasjoner står fritt til å velge om brukere skal belastes eller ikke belastes når applikasjonen brukes, og enhver Facebook-bruker har mulighet til å bli en tredjepartsutvikler og lage slike applikasjoner. Hver applikasjon blir tildelt en assosiert applikasjonsside i Facebook, som med gruppe- og fan-sidene. Denne siden viser informasjon om applikasjonen, som blant annet applikasjonens utvikler, antall brukere av applikasjonen, applikasjonens formål og funksjonalitet, og en link til selve applikasjonen. Applikasjoner kan oppholde seg i Facebook-domenet eller lede til en ekstern webside utenfor Facebook-domenet. Det er også støtte for utvikling av desktop-

applikasjoner, og mobile applikasjoner.

4.2.3 Stedstjenester

Facebook Places er en stedstjeneste Facebook tilbyr, som tillater brukere å dele steder de liker. Brukere av applikasjonen har mulighet til å dele steder ved å *sjekke inn* til stedet, og la andre Facebook-venner se hvor brukeren befinner seg for øyeblikket [43]. Alle innsjekkede lokasjoner representeres med en egen side i Facebook som for applikasjoner, og inneholder detaljert informasjon om lokasjonen og innsjekkinger som er blitt gjort av andre brukere. Brukere kan legge til lokasjoner dersom det ikke finnes en side til lokasjonen fra før. Brukeren kan også sette en «tag» på venner som er med, og publisere en beskjed sammen med innsjekkingen for å fortelle mer om den. Med applikasjonen kan man også se om noen andre venner har sjekket inn i nærheten. Facebook Places er tilgjengelig gjennom *Facebook for iPhone*- eller *Facebook for Android*-applikasjonen, eller ved å logge inn til applikasjonens smartphone-side.

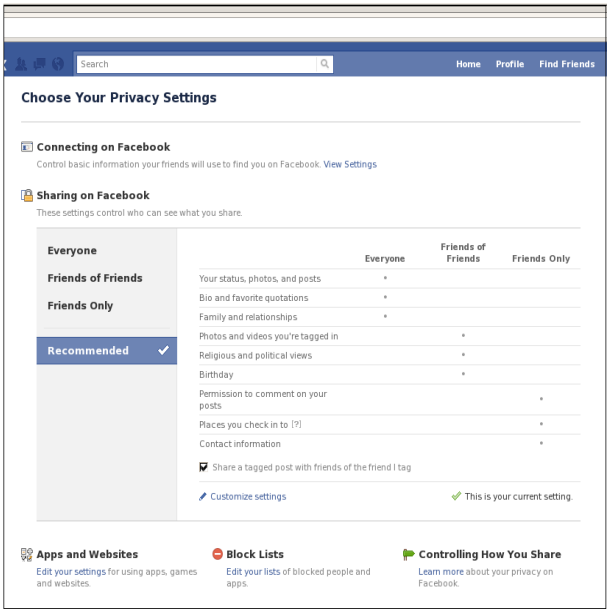
4.3 Personvern

Personvern i Facebook har vært et omdiskutert tema, helt siden nettstedet ble grunnlagt. Selv om formålet med Facebook er «å gjøre verden mer åpen», så er det også viktig å beskytte brukeres personvern, og tillate brukerne å ha kontroll over flyten av deres personlige informasjon.

Første versjon av Facebook var forholdsvis enkel med svært begrenset funksjonalitet. Det fantes ingen bilde og video-deling eller mekanisme for meldingsutveksling. Bruksområdet var innskrenket til studenter ved universiteter i USA [69], hvor brukere kunne bli medlem av forskjellige nettverk, for å fortelle hvilken region eller universitet de hadde tilhørighet til. Nettstedets standard var på den tid slik at alle brukere fikk innsyn i grunnleggende informasjon om andre brukere, og all annen informasjon var kun synlig for venner og brukere som tilhørte samme nettverk som brukeren selv.

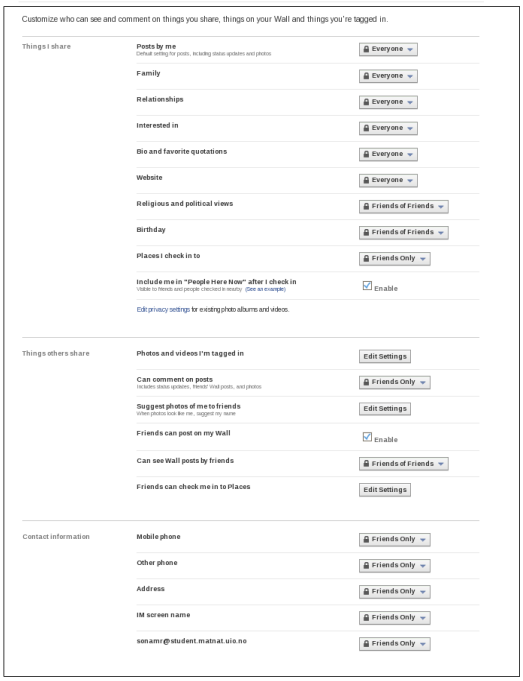
Etter hvert som Facebook har vokst frem til å tilby større og mer kompleks funksjonalitet, så har personvern og kontroll av informasjonsflyt blitt et viktigere tema. Det sosiale nettstedet har i dag utformet en ny modell for personvern med et eget skriv om vilkår for personvern som skal være lengre enn den amerikanske grunnloven [64]. Regionale nettverk er vokst til å inkludere flere og flere mennesker, slik at standard innsyn i brukerprofiler i samme nettverk er blitt luket bort. For å ha oversikt på individnivå, så kan Facebook-brukere opprette egendefinerte vennelister, som har spesifisert tilgang og innsyn i brukerens profil. Brukeren kan befolke listene, og deretter begrense listene etter forskjellige kriterier. Brukeren kan eksem-

pelvis sette at den ene listen ikke skal ha innsyn i bilder og videoer, og den andre listen ikke skal ha innsyn i bilder, videoer og innlegg som publiseres på brukerens vegg. Brukeren kan redigere listene og listenes restriksjoner til enhver tid. Brukeren kan også kontrollere hvordan profilen skal se ut for venner av venner, og Facebook-brukere som ikke er i relasjon med brukeren selv.



Figur 4.4: Facebook personverninnstillinger.

Informasjonsflyten deles inn i tre kategorier: informasjon brukeren deler, informasjon andre deler med brukeren, og brukerens kontaktinformasjon. De tre kategoriene er igjen inndelt i noen underkategorier, for å ha så detaljert kontroll som mulig:



Figur 4.5: Detaljert personverninnstillinger.

Ettersom Facebook er blitt en populær arena for tredjepartsutviklere, så har slike tredjepartsutviklede applikasjoner som standard samme type restriksjoner som brukere har satt på Facebook-brukere som ikke er i relasjon med brukeren selv. Hvordan denne restriksjonen er satt, varierer fra bruker til bruker. Dersom brukeren har lagt til tredjepartsapplikasjonen til din brukerprofil og dermed er en applikasjonsbruker, så har applikasjonen som standard kun innsyn i brukerens grunnleggende informasjon. Dersom applikasjoner ønsker å aksessere utvidet informasjon om brukeren, må tillatelse forespørres på en spesifikk måte. Applikasjonsutvikleren må angi nøyaktig hvilke data hun er interessert i å aksessere i applikasjonen, og en innvilgelse må gis fra brukeren før dataene kan aksesseres av applikasjonen. Dette gjør at det ikke er en «alt eller ingenting»-strategi på informasjonsdeling fra Facebook sin side, men gir brukeren detaljert kontroll på dataene sine. Innvilgede aksesser til applikasjoner kan oppheves av brukeren til enhver tid.

Gruppe- og fan-sider kan også hemmeligholdes slik at et utvalg av brukere kan «se» sidene. For utenforstående brukere ville det vært som om gruppen ikke eksisterer. Man kan også innstille grupper til å være *lukket*, slik at brukere må inviteres til å bli med i gruppen, i motsetning til *åpne* grupper, som alle kan melde seg inn i. Hendelser i hendelsessider kan også usynliggjøres av verten. Verten kan spesifisere om alle kan komme til begi-

venheten eller om man trenger en invitasjon. Dersom man trenger en invitasjon til hendelsen, vil som regel kun brukere med invitasjon ha innsyn i hendelsessidens eksistens.

Som standard kan en Facebook-bruker sende private beskjeder til alle andre Facebook-brukere, selv om de to brukerne ikke skulle vært i en relasjon på Facebook. I tilfeller hvor en bruker mottar uønskede beskjeder fra uvedkommende, har brukeren muligheten til å *blokkere* eller å *rapportere* den uvedkommende. Ved blokkering vil brukerens profil permanent skjules for den uvedkommende, slik at brukeren ikke lenger kan kontaktes av den uvedkommende. Dersom en bruker rapporterer en annen bruker, så vil også den rapporterte brukeren miste synlighet over den rapporterende brukerens profil, og den rapporterte brukeren vil i tillegg få en bemerkning hos Facebook. Dersom en bruker blir tildelt et visst antall bemerkninger, så slettes brukerkontoen til brukeren permanent fra Facebook.

4.4 Inntekt og finansiering

Å være medlem av Facebook er en gratis tjeneste for Facebook-brukere, imidlertid har Facebook store årlige inntekter. I 2006 var fortjenesten anslått til 52 millioner amerikanske dollar. I det påfølgende året hadde Facebook sin største økning på 188 % som tilsvarer 150 millioner $ i omsetning. I 2008 omsatte nettstedet 280 millioner $, og i 2009 lå den årlige fortjenesten på 800 millioner $. Inntektene i 2010 ble beregnet til å være på 2 milliarder $.[70].

Den største inntekten til Facebook ligger i *Credits* og *Flyers*-applikasjonene utviklet av Facebook-utviklere. Facebook Credits er en virtuell kurs som Facebook-brukere kan bruke til å kjøpe virtuelle goder i spill og applikasjoner i Facebook-plattformen. For èn amerikansk dollar mottar brukeren 10 credits [71]. Facebook beholder 30 prosent av all fortjeneste opptjent gjennom Credits-applikasjonen.

Facebook Flyers er et internt annonseringssystem i Facebook-domenet som eksterne annonsører kan benytte seg av. Med dette systemet kan annonsører rette annonsene sine mot målbrukere basert på spesifikk data fra brukerprofilene, som interesser, alder, region, utdanning og sivil status [40]. Store internasjonale selskaper avhenger av annonsering via Facebook for å opprettholde en økning i salg, ettersom Facebook tilbyr over 600 millioner potensielle kunder [79]. Facebook er i dag det største forumet for fremvisningsannonsering, da Facebook viser frem 50 milliarder annonser i gjennomsnitt per måned [79].

Kapittel 5

Facebook-plattformen

Som vi har sett hittil, så er Facebook ikke bare en sosialiseringstjeneste for forbrukere, men Facebook er også en plattform som tillater tredjepartsutviklere å bygge egne applikasjoner. Nettstedet har oppnådd en enorm nettverkseffekt ettersom man kan nå i overkant av 600 millioner brukere, deriblant familie, venner, bekjente, og kolleger. Facebook er dermed en fin arena for å tilby et oppslagsverk av SIP-adresser, og å tilgjengeliggjøre VoIP-tjenester. I de neste kapitlene skal vi se på Facebook sin infrastruktur, for å forstå hvordan plattformen fungerer. Videre skal også ta en titt på plattform-APIene og programvareutviklingsverktøy som tilbys av Facebook, for å se på mulighetene man har for å lage en integrasjon mellom Facebook og VoIP-tjenester.

5.1 Facebook infrastruktur

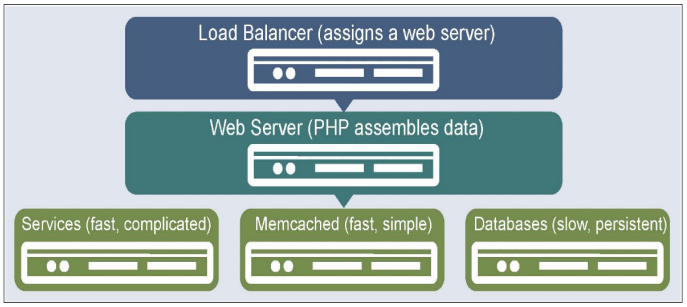
På den skalaen Facebook opererer, så vil de fleste tradisjonelle tilnærmingene på å servere web-innhold, brytes ned eller helt enkelt ikke være praktisk. Før vi ser på hvordan Facebook er bygd opp, så ser vi på en oversikt over skaleringsutfordringene nettstedet står ovenfor. Tallene er hentet fra [42]:

- Facebook serverer 570 milliarder sidevisninger i måneden
- Det er flere bilder på Facebook enn i andre populære bildedelingssteder kombinert (inkludert sider som *Photo Bucket* og *Flikr*)
- Mer enn 3 milliarder bilder lastet opp i Facebook i måneden
- Facebooks systemer serverer i gjennomsnitt 1.2 millioner bilder i sekundet [1]
- Over 30 milliarder informasjonsbiter [2] deles hver måned
- Over 130 Terabyte med informasjon logges hver dag

[1] Inkluderer ikke bilder som serveres via Facebook sitt CDN (Content Delivery Network)

[2] Eksempler på informasjonsbiter kan være statusoppdateringer, kommentarer, notater, linker, bilder, videoer og lignende.

Facebook er i hovedsak blitt utviklet fra bunnen og opp med åpen kildekode programvare, og bruker en variasjon av tjenester, verktøy og programmeringsspråk i implementasjonen. Det er fire hovedkomponenter som utgjør kjernen i infrastrukturen til Facebook, figuren er tatt fra [11]:



Figur 5.1: Facebooks kjernekomponenter i infrastrukturen.

Øverst finnes en *Load Balancer* som tar i mot forespørsler fra Facebook.com og tildeler forespørsler til hver av Facebooks webservere, for prosessering. Med lastbalansereren blir dermed forespørslene håndtert på en belastningsbalansert måte. Videre så er den første kjernekomponenten i Facebook sin infrastruktur, *Facebook webservere* eller *Facebook dataservere*. På frontsiden[3] av Facebooks webservere så kjøres en *LAMP (Linux, Apache, MySQL, PHP)* stakk. På bakenden[4] så er tjenestene skrevet i forskjellige programmeringsspråk, som C++, Java, Python og Erlang. Webserverne selv er skrevet i et optimert PHP-språk, utviklet av Facebook-ingeniører. Den optimerte PHP-varianten er forbedret til å være mer CPU- og minne-effektiv for å tilpasse nettstedets skaleringsutfordringer, og gjør det også enklere å skrive utvidelser (extentions). En av utvidelsene som Facebook-ingeniørene har utviklet ut i fra den optimerte PHP-varianten, er, *Hip HoP for PHP*. Hip Hop for PHP er en kildekode transformator som er utviklet for å spare serverressurser, ettersom skriptkode og er relativt tregere enn maskinkode. Transformatoren tar i mot PHP kildekode og gjør den om til optimert C++ kode og bruker deretter g++ til å kompilere til maskinkode, for bedre ytelse [4, 11].

Facebook webservere interagerer med en annen viktig kjernekomponent, *Services*. I denne tjenestekomponenten så bygges og tilbys bakside-tjenester til webserveren. Filosofien er å generere nye tjenester kun ved

[3]Frontsiden av en server er et grensesnittet mellom brukeren og bak-enden. Frontsiden skal ta i mot forespørsler fra brukeren i forskjellige format, og gjøre dem om slik at de er i overensstemmelse med en spesifikasjon bakenden kan bruke.

[4]Bakenden av en server utfører som regel interne prosesser, og har ingen dirkete kommunikasjon med brukere eller eksterne prosesser. Bakenden kommuniserer med «utsiden» via frontsiden.

behov, lage enklest rammeverk for opprettelsen av tjenesten, og å bruke riktig programmeringsspråk som er egnet for oppgaven [4]. Over tjenestemodulen bruker Facebook-servere et *Thrift*-interface. Thrift er et rammeverk for fjernprosedyrekall og brukes til utvikling av skalerbare tjenester på tvers av programmeringsspråk. Rammeverket støtter blant annet C++, PHP, Python, Perl, Erlang, Ruby og Java [63]. Thrift er raskt og sparer utviklingstid, og tilbyr dessuten også oppdelinger av arbeid på høytytelses servere og applikasjoner [4]. Thrift-verktøyet egner seg for systemer som benytter tjenester som er skrevet i mange forskjellige språk, slik som Facebook. Webserverne bruker også verktøyet, *Scribe*. Scribe brukes for å aggregere loggdata strømmet i sanntid fra et stort antall servere [63]. Det er et skalerbart rammeverk for å logge store mengder av data, og kan bygges på toppen av Thrift. Scribe brukes i Facebook til å flytte data fra webserverne til sentrale lagringsenheter, og håndterer automatisk nye loggkategorier ettersom de dukker opp [5]. Facebook håndterer mer enn 130 Terabyte med loggdata daglig, som er tusen ganger volumet av post som leveres med posttjenester i USA. Scribe er derfor en viktig del av datakomponenten i infrastrukturen til Facebook.[32].

Facebook bruker det optimaliserte memory-caching systemet *Memcached* [4], som et caching-lag mellom webservere og MySQL-servere. Memcached er Facebooks primære form for caching, og lindrer databasebelastningen. Memcached er implementert som en distribuert hashtabell som er holdt i webservernes minne. Cachene kan lagre en variasjon av data, alt fra resultater av MySQL-spørringer til generisk data som legges manuelt inn i cachene av Facebook-ingeniørene. Cachene er relativt enkle, da de kun har 5 operasjoner å utføre på dataene: *Get*, *Set*, *Increment*, *Delete* og *Multiget/Multiset*. Imidlertid så er cachene meget raske ettersom de kan utføre 150 millioner operasjoner i sekundet [11]. Facebook har gjennom årene gjort en rekke optimaliseringer av Memcached-programvaren [42]. Facebook kjører tusenvis av Memcached servere med et titalls Terabyte av data cachet til enhver tid, og er den største Memcached-installasjonen som finnes i dag. Ved å ha et slikt caching-system så kan Facebook servere data til brukere enda raskere. En ulempe med Memcached er at cachingen skjer *out-of-band*[6]. Dette medfører at cache-visninger ikke nødvendigvis er konsistent og gir samme visning som MySQL-databasen. Facebook opprettholder koherensen mellom cache og database på egenhånd.

Den siste kjernekomponenten i Facebook sin infrastruktur er *MySQL-databasene*. Over databasene har Facebook-ingeniører skrevet APIer som gjør det mulig å se på databasen som en graf med objekter, og linker mellom disse objektene. I praksis så vil MySQL-databasen være et persistent lagringsmedium, Memcached vil være en distribuert indeks, og webser-

[5] Facebook har et hundretalls logg-kategorier.

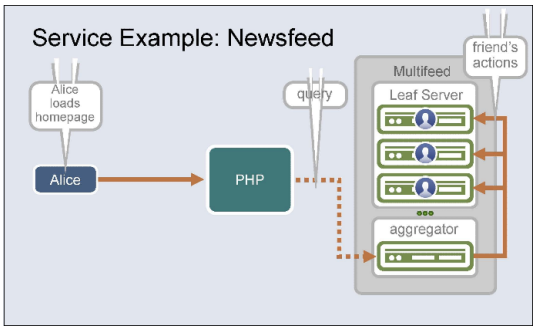
[6] At data sendes out-of-band betyr at dataene sendes i en separert datastrøm.

verne slår sammen og prosesserer data fra cachene og databasene [11]. La oss videre se på et eksempel, for å illustrere dataflyt i Facebook.

Når en bruker logger inn på sin Facebook-konto, vil hun først bli presentert med nyhetsstrømmen fra brukerens hjemside:

Figur 5.2: Facebook nyhetsstrøm i hjemsiden.

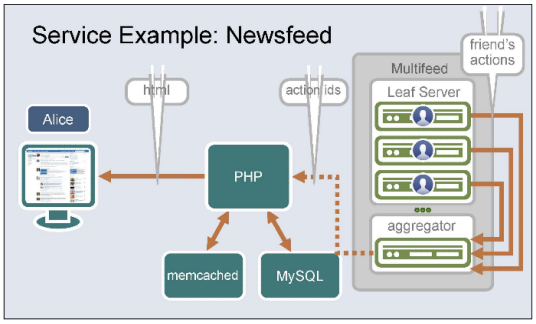
Nyhetsstrømmen skal bestå av de ferskeste og mest interessante nyhetene om brukerens venne- og side-aktiviteter, som endrer seg hele tiden. Altså dersom brukeren eksempelvis logger inn i Facebook kl.19.00 og så igjen kl.20.00 en gitt dag, er det ønskelig at det vises en annen og nyere nyhetsstrøm kl.20.00 enn den som ble vist ved tidligere innlogging. Når brukeren logger inn, utføres det en tjeneste i tjenestemodulen, som skal hente ny nyhetsstrøm:



Figur 5.3: Facebook Innlogging: Innlasting av nyhetsstrøm

Forespørselen om denne tjenesten går fra nettleseren i form av HTML-kode til en av Facebook sine PHP-kodede webservere. Webserveren gjør så videre et kall om tjenesten på *leaf servers* (bladservere) som opprettholdes av Facebook for å cache nye og interessante hendelser som har hendt brukere. Bladserverne inneholder en *multifeed* og en *aggregator*. Multifeed er en ettnivås hashmap som holder på dataene. Mappen har en ID for hver bruker og en liste over nye hendelser rundt brukeren. Slike bladservere kommer i clustere[7], og hvert slikt blad-cluster har flere duplikater for at tjenesten skal skalere. Aggregatoren brukes til å utføre spørringer på multifeed'en. I dette tilfellet vil webserveren utføre spørringer over Thrift mot aggregatoren til å filtrere ut nyheter på venner av brukeren. ID-ene til data som tilfredsstiller spørringen vil returneres til webserveren.

I webserveren så finnes nå kun hendelses-IDene, men for en komplett publisering på brukerens hjemside må også metadataene til nyhetene hentes. Slik metadata kan være aktør, assosiert media, tidsstempel, kommentarer, og lignende:

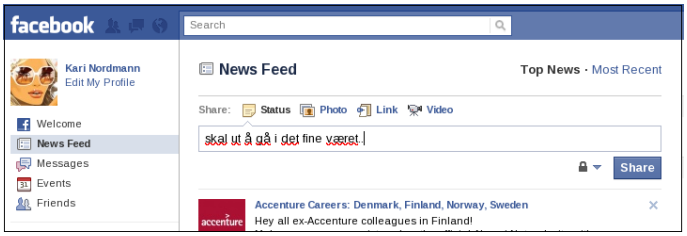


Figur 5.4: Facebook Innlogging: Innlasting av nyhetsstrøm

[7]Server-clustere er en gruppe av linkede server som jobber tett sammen, og på mange måter former en enkel server.

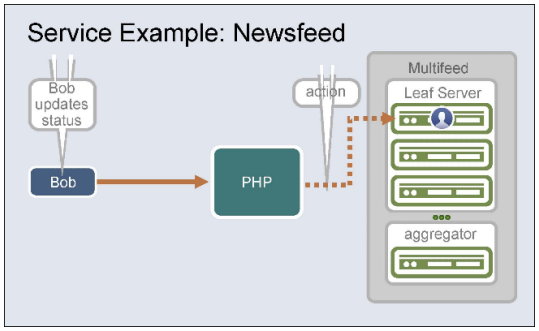
For å hente metadata, gjør webserveren kall mot en Memcached-server i samme cluster som bladserveren, om å returnere IDene til metadata assosiert med hendelses-IDene. Dersom ønsket data ikke ligger i cachen, gjøres kallet automatisk videre til en MySQL-server som også er i samme cluster. Etter at metadata er returnert til webserveren, vil det utføres en assemblering av all mottatt data, og innholdet vil bli gjort om til HTML-kode. Deretter returnerer webserveren nyhetsstrømmen til brukeren på hjemsiden i brukerens profil.

I et annet eksempel så antar vi at brukeren har logget inn i Facebook-kontoen sin, og fått ny nyhetsstrøm. Brukeren ønsker så å oppdatere statusen sin:



Figur 5.5: Facebook statusoppdatering.

På samme måte som eksempelet ovenfor, så vil forespørselen gå fra nettleseren til en Facebook webserver. Webserveren vil så gjøre et kall over Scribe mot en bladserver. I multifeed'en i bladserveren vil denne statusoppdateringen med sin hendelses-ID logges inn i listen med brukerens ID sammen med andre oppføringer om brukerens nyligste aktiviteter og hendelser:



Figur 5.6: Facebook statusoppdatering logges.

Det er mer enn 50 milliarder bilder som skal lagres i Facebook, og hvert bilde skal lagres i 4 forskjellige oppløsninger: Thumbnail-versjon, album-

versjon, normal størrelse og stor størrelse. Dette gir en total på 200 milliarder bilder [42]. Bildehåndteringssystemet til Facebook må derfor ikke bare være i stand til å håndtere slike store mengder med data, ytelsen er også kritisk ettersom Facebook skal fremvise 1.2 millioner bilder i sekundet. *HayStack* er Facebook sitt høytytende bildelagrings- og opphentingssystem. HayStack fungerer som et *overlay* filsystem hvor i stedet for at hvert bilde er lagret i en fil, så er flere bilder organisert i *blobber* [8], og hver blob lagres i èn fil. Hvert bilde med en gitt bilde-ID ligger på et kjent offset i filen og kan enkelt opphentes.

Facebook har i dag ansatt omtrent 2000 programmerere og dataingeniører, som hver håndterer data for mer enn 600 millioner brukere[9]. Per September 2010 serverte Facebook innhold fra over 60 000 servere som en blanding av Memcached-servere, web-servere og database-servere [32]. Serverne er linket til utsiden via fiberoptiske kabler. Dette plasserer nettstedet på fjerde plass av høyest antall servere, blant de største Internett-selskapene i verden, hvor *Intel* er på første plass med 100 000 servere. Facebook opererer for øyeblikket i fra ni datasentre lokalisert i USA, både langs vestkysten og østkysten. Datasentrene utgjør til sammen mellom 10 000 - 35 000 kvadratmeter. Facebook-teamet er i gang med å ekspandere ytterligere, og jobber i dag med å bygge sitt eget datasenter. Datasenteret skal ha blitt estimert til å være på 147 000 kvadratmeter [33].

Hittil har vi sett på infrastrukturen til Facebook, for å forstå hvordan datainnhold serveres og lagres. I den neste seksjonen skal vi ta en nærmere titt på det mest sentrale APIet Facebook tilbyr tredjepartsutviklere for å utvikle egne applikasjoner. For mer om infrastrukturen til Facebook, se [32, 9, 30, 33, 11]

5.2 Facebooks Graph API

I løpet av 2010 har Facebook-plattformen gjennomgått større forandringer som påvirker hvordan utviklere kan integrere med nettstedet. Facebook APIet brukte tidligere et *REST*-lignende interface. Alle metodekall som blir gjort mot Facebook APIet gjøres i form av enkle *HTTP GET* og *HTTP POST* anmodninger over Internett, og videre til Facebook-servere. Det ble tidligere brukt en REST API server for å returnere informasjon fra API kallene. [48].

Det nyeste APIet som Facebook tilbyr, kalles *Facebook Graph API*. Formålet med å innføre graf-APIet er å drastisk forenkle måten utviklere leser og skriver data i Facebook [58]. Graf-APIet skal gi en enkel og konsistent oversikt over Facebook i form av en sosial *graf-struktur*. Facebook-grafen er representert av *objekter* (i form av brukere, publisert innhold, applikasjonssider og lignende) og *forbindelsene* mellom disse (relasjoner, tags, kommen-

[8] En blob er en samling av binær data som ligger lagret som en singel entitet

tarer og annet felles innhold). Hvert objekt har en unik ID, og data assosiert med objektet kan hentes ved å kalle:

```
https://graph.facebook.com/ID
```

For eksempel så har den offisielle siden for Facebook-plattformen, ID = 19292868552. Ved å kalle *https://graph.facebook.com/19292868552* så returneres meta-informasjon tilknyttet objektet:

Figur 5.7: Graf-API objekt.

Dersom objektet har et brukernavn, så kan assosiert informasjon til objektet også returneres ved å kalle med brukernavnet i stedet for ID-en:

```
https://graph.facebook.com/brukernavn
```

I eksempelet ovenfor så ser vi av den returnerte informasjonen at Facebook-plattformen har brukernavnet, *platform*. Ved å kalle *https://graph.facebook.com/platform* ville returnert det samme innholdet som i kallet ovenfor. Dersom utvikleren ønsker å hente informasjon om forbindelser mellom objekter, så kan det oppnås med å gi forbindelsestypen bak objektet:

```
https://graph.facebook.com/<ID/brukernavn>/forbindelse_type
```

ID eller brukernavn angir hvilket objekt man ønsker å returnere forbindelser til, og forbindelse_type angir hvordan type objektforbindelse man vil hente fra grafen. Ved å eksempelvis kalle *https:facebook.com/me/friends* så returneres alle venner av brukeren selv, da objektet «me» er alias til brukeren selv. «Friends» angir at man ønsker å se alle vennerelasjonene brukeren har til sin brukerprofil.

Følgende objekter er definert i graf-APIet. Datahenting gjøres med HTTP GET forespørsler:

Objekt type	Beskrivelse
Album	Et foto album
Application	En applikasjon registrert i Facebook
Checkin	En innsjekking gjort via Facebook Places
Comment	En kommentar på et graf-API objekt
Event	En Facebook begivenhet
Friendlist	En Facebook venneliste
Group	En Facebook gruppe
Insights	Statistikk om applikasjoner, sider og domener
Link	En delt link
Message	En beskjed i Facebook sitt meldingssystem
Note	Et Facebook notat
Page	En Facebook side
Photo	Et indivuelt bilde
Post	En oppføring i profilens nyhetsstrøm
Review	En anmeldelse av en applikasjon
Status Message	Status beskjed i en brukers profil
Subscription	En abonnering fra en applikasjon
Thread	En meldingstråd i Facebook sitt meldingssystem
User	En bruker-profil
Video	En individuell video

Tabell 5.1: Oversikt over objekttyper i graf-APIet.

Hvert av objektene har et sett felter med informasjon tilknyttet objektet, og forskjellige type forbindelser som assosieres med objektet. Slike attributter og forbindelser varierer fra objekt til objekt. Som regel returneres alle av objektets felter når man gjør et kall på objektet i graf-APIet. Utvikleren kan filtrere ut kun de feltene hun er interessert i ved å legge til *fields*-parameteren i slutten av spørringen. I kallet nedenfor så hentes det kun ut navnet og profilbildet til venner av brukeren:

```
https:facebook.com/me/friends?fields=name,picture
```

Når man utfører spørringer på forbindelser til et objekt, kan dataene filtreres med *limit*-, *offset*-, *since*- og *until*-parametere. Limit/Offset-betingelsene begrenser mengden av det returnerte resultatet, og Until/Since-betingelsen fungerer som et tidsstempel fra tidsperioden søket skal gjelde.

Videre kan utviklere publisere til Facebook på flere måter via graf-APIet. Følgende tabell viser hvilke objekter man kan publisere på, og hvilke publiseringsmetoder de støtter. Publiseringen gjøres med HTTP POST forespørsel:

Metode	Beskrivelse
PROFIL_ID/feed	skrive til gitt profils vegg
OBJECT_ID/comments	kommentere et gitt objekt
OBJECT_ID/likes	like et gitt objekt
PROFILE_ID/notes	publisere notat til gitt profil
PROFILE_ID/links	publisere link til gitt profil
PROFILE_ID/events	opprette en begivenhet
EVENT_ID/<attending/maby/decline>	svare på begivenhet
PROFIL_ID/albums	opprette et fotoalbum
ALBUM_ID/photos	laste opp bilde til gitt fotoalbum
PROFILE_ID/checkins	opprette en innsjekking til en lokasjon representert av en side

Tabell 5.2: publisering med Graph API

For eksempel kan man publisere en kommentar til gitte objekt med objekt-ID, «object_id», og beskjeden, «din_beskjed»:

```
https://graph.facebook.com/object_id/comments?message=din_beskjed
```

Dersom man ønsker å slette en kommentar til et objekt, kan man utføre en HTTP DELETE forespørselen. Ettersom hver kommentar har en egen ID kan man slette kommentaren med kommentar-ID, «comment_id», med post-ID, ved å legge til *method=delete* bak posten:

```
https://graph.facebook.com/comment_id?method=delete
```

Når tredjepartsutviklere oppretter og registrerer sin applikasjon i Facebook, får hun en detaljert analyse over demografien til applikasjonensbrukerne, og hvordan applikasjonsbrukerne deler informasjon. Slik analytisk data kan utvikleren hente ved hjelp av *Insight*-objektet. Insight-data kan også integreres i utvikleres egendefinerte analysesystem, og kan hentes ut slik:

```
https://graph.facebook.com/applikasjons_id/insights
```

Spørringen gir ut analytisk data tilgjengelig om applikasjonen med ID, «applikasjons_id», som finnes i graf-APIet. Eksempel på slik analytisk data kan være totalt antall applikasjonsbrukere, antall aktive applikasjonsbrukere, og en rekke andre beregninger og statistikker om applikasjonen.

Facebook anbefaler sine tredjepartsutviklere å bruke graf-APIet med HTTP metodekall når de ønsker å hente eller publisere informasjon i sine Facebook-applikasjoner. Imidlertid tilbyr Facebook også andre verktøy for mer avansert datahenting, og for applikasjoner som befinner seg utenfor Facebook-domenet. I neste kapittel skal vi ta en nærmere titt på disse avanserte APIene og programvareutviklingsverktøy (SDKer) som tilbys av Facebook. For mer om Facebooks sitt graf API, se [54, 48, 58].

Kapittel 6

Avanserte APIer og SDKer

6.1 Facebook Query Language

Facebook Query Language, eller *FQL*, er et spørrespråk utviklet av Facebook for å tillate tredjepartsutviklere å interagere med Facebook-plattformen. FQL lar utviklere bruke et *Structured Query Language (SQL)*-syntaktisk grensesnitt, for å utføre spørringer på data som finnes i graf-APIet. FQL tilbyr avanserte funksjoner som ikke er tilgjengelig gjennom metodekall på graf-APIet, for eksempel som å flette flere spørringer i ett enkelt kall [57]. Utviklere kan spørre med FQL på følgende måte:

```
https://api.facebook.com/method/fql.query?query=din_spørring
```

Utvikleren kan bestemme repsonsformatet på spørringen med parameteren, *format*, som returnerer responsen som *XML* eller *JavaScript Object Notation (JSON)*. FQL-spørringer er på formen:

```
SELECT [felter] FROM [tabell] WHERE [betingelse]
```

Til forskjell fra SQL, så kan FROM-klausulen kun inneholde èn tabell. Utvikleren kan bruke *IN* nøkkelordet i SELECT-klausuler eller en *WHERE*-klausul, for nøstede spørringer. Nøstede spørringer kan ikke referere til variable i ytre spørringsskop. Spørringene må være *indekserbare*, som betyr at spørringen må spørre på tabeller som er definert av Facebook, og spørringene må spørre etter indekserbare attributter, som er attributter Facebook har definert og assosiert til hver tabell. Følgende tabell gir en oversikt over noen av de indekserte tabellene Facebook har definert, og noen av deres tilhørende indekserte attributter. For fullstendig beskrivelse av tabellene og attributtene, se [57]:

Indekserte tabeller	Indekserte attributter	Beskrivelse
album	albumID, owner, cover_pic_id	informasjon om album
application	app_id, api_key, app_name	informasjon om applikasjon
comment	xid, comment_id, post_id	kommentar evt. assosiert med nyhetsstrøm
connection	source_id	brukerens venner og tilkoblede sider
cookies	uid	cookie informasjon
developer	developer_id	applikasjoner til en utvikler
family	profile_id	informasjon om brukers familie
friend	uid1, uid2	sjekke om to brukere er forbundet
friendlist	owner, flid	vennelister til en bruker
friendlist_member	flid, uid	medlemmer av en venneliste
friend_request	to_uid	sendte/mottatte venneforespørseler for en bruker
insights	object_id	analytisk data om applikasjon eller side
like	object_id, post_id	brukere som liker et gitt objekt
link	link_id, owner	publiserte linker til en bruker
link_stat	url	detaljert info om utviklerens implementasjon
mailbox_folder	folder_id, viewer_id	informasjon om brukers mailboks
message	message_id, thread_id	informasjon om beskjed i en tråd
metrics	end_time, period	beregninger av utviklers applikasjon
note	uid	publiserte notater til en bruker
notification	recipient_id	varslinger til en bruker i nåværende sesjon
page	page_id, name	informasjon om en Facebook side
page_admin	uid	informasjon om administrasjon av en Facebook side
permissions	uid	de utvidede tillatelsene brukeren har gitt applikasjonen
permissions_info	permission_name	informasjon om en tillatelse
photo	photo_id, album_id	informasjon om et bilde
privacy	object_id	brukerens personverninnstillinger til gitt objekt
profile	id	offentlig info om en brukers profil eller en Facebook side
standard_friend_info	uid1, uid2	om to brukere er linket som venner
standard_user_info	uid, name	informasjon om bruker til analytisk formål
status	uid	statuser til en bruker
stream	post_id, src_id, filter_key	brukeren nyhetsstrøm
stream_filter	uid	oppnå en filter_key
thread	thread_id, folder_id	informasjon om beskjed-tråder i en mappe
translation	locale, native_hash, pre_hash_string	opprinnelig og oversatt tekst
user	uid, name	detaljert informasjon om en bruker
video	video_id, owner	informasjon om en video

Tabell 6.1: FQL tabeller og attributter

For eksempel kan utvikleren på følgende måte hente navnet på vennelistene som bruker med Facebook-ID, id1, er medlem av:

```
SELECT name FROM friendlist WHERE flid IN
(SELECT flid FROM friendlist_member WHERE uid = "id1")
```

Ved å bruke FQL til å slå sammen spørringer, unngås flere «roundtrips» fra utviklerens server til Facebooks servere. Facebooks servere kan noen ganger utføre en optimering, slik at dersom data brukes i multiple spørringer, så caches dataene. For mer om FQL, se [57].

6.2 JavaScript SDK

*JavaScript SDK*et er et programvareutviklingsverktøy som tilbys av Facebook, og som tillater utviklere å aksessere graf-API funksjoner og Dialoger via JavaScript. JavaScript SDKet tilbyr også et rikt sett av *client-side*[1] funksjonalitet for autentisering og autorisasjon [60]. Facebook tilbyr også en JavaScript testkonsoll, som lar utviklere teste og debugge SDK-metodene. SDKet har åpen kildekode og er tilgjengelig for utviklere å laste inn i siden sin, men for å benytte JavaScript SDKet må siden ha en applikasjons-ID. Alle

[1]Client-side referer til operasjoner som typisk utføres av klienten i en klient-server forhold.

eksterne nettsider og applikasjonssider tildeles en slik applikasjons-ID når de integrerer med Facebook-plattformen.

JavaScript SDKet lastes inn i et dokument ved å bruke det standarde «script»-elementet. Metoden «window.fbAsyncInit()» er den første metoden som kalles etter at JavaScript SDKet er lastet inn i dokumentet. Metoden kaller først på en annen metode, «FB.init()». FB.init() tar seg av initialiseringen av SDKet. Init-metoden tar blant annet applikasjons-IDen, og en rekke parametere som blant annet bestemmer om innloggingstilstanden skal sjekkes, og om cookies skal settes slik at serveren kan aksessere sesjonen. Videre så lager fbAsyncInit()-metoden en forekomst av elementet, og inkluderer JavaScript SDKet. Metoden laster inn JavaScript SDKet asynkront, for å ikke blokkere andre elementer. All annen JavaScript-kode som man ønsker å kjøre etter SDKet er blitt lastet inn, kan plasseres innenfor fbAsyncInit-metoden. Et eksempel på en innlasting og initialisering av JavaScript SDKet er slik:

```
<script>
  window.fbAsyncInit = function() {
    FB.init({appId: 'din_appID', status: true, cookie: true,
             xfbml: true});
  };
  (function() {
    var e = document.createElement('script'); e.async = true;
    e.src = document.location.protocol +
      '//connect.facebook.net/en_US/all.js';
    document.getElementById('fb-root').appendChild(e);

    ...

  }());
</script>
```

Utviklere kan hente data fra Facebook graf-APIet, eller sende inn data til Facebook, ved bruk av JavaScript SDKet. For å utføre graf-metodekall med JavaScript SDKet, brukes metoden, «FB.api()». Metoden tar to argumenter, hvor det første argumentet er graf-objektet eller objekt-forbindelsen med tilhørende spørrebetingelser, det andre argumentet er en funksjon som tar i mot responsen av api-kallet. I eksempelet nedenfor så gjøres et kall på bruker-objeketet, og i responsen skrives objektets navn ut i dokumentet:

```
FB.api('/me', function(response) {
  alert(response.name);
});
```

Med JavaScript SDKet kan utviklere av tredjepartsapplikasjoner også logge inn brukere i applikasjoner og eksterne nettsider, og utføre autentisering og autorisasjon av brukerne[53]. For å logge brukeren inn i siden, så må Facebook autentisere brukeren for å sørge for at brukeren er den han påstår han er. Deretter må Facebook autentisere den eksterne nettsiden eller applikasjonssiden. Dette sørger for at brukeren gir sin informasjon til utviklerens side, og ingen annen side. Til slutt må brukeren eksplisitt autorisere applikasjonen med aksess til brukerens informasjon. Dette sørger for at brukeren vet nøyaktig hva slags personlig informasjon han meddeler siden.

Prosessen av brukerautentisering, applikasjonsautentisering og applikasjonsautorisasjon håndteres imidlertid av SDKet. Utvikleren trenger kun å implementere en *Login Button* eller innloggingsknapp via JavaScript SDKet. En slik innloggingsknapp er en plugin som gir mulighet til besøkede

av eksterne nettsider eller tredjepartsapplikasjoner til å logge inn i Facebook, med sin Facebook-identitet. I det brukeren trykker på innloggingsknappen blir han spurt om han tillater den siden han logger inn via (ekstern nettside eller applikasjonsside), aksess til Facebook-profilen hans. Hvilke aksesser siden ønsker, kan spesifiseres av utvikleren med plugin-attributtet, «perms». Knappen viser profilbildet til brukeren etter en suksessiv innlogging. Utvikleren kan tilby å vise profilbilder av venner av brukeren, som allerede er logget inn i siden. Kode for en innloggingsknapp med JavaScript SDKet ser slik:

```
<fb:login-button> evt_annen_tekst </fb:login-button>
```

Knappen har som standard teksten *Login*, dersom utvikleren ønsker å ha annen tekst på knappen, kan det angis i stedet for «evt_annen_tekst» i kodesnutten ovenfor. Innloggingsknappen har også følgende attributter:

show_faces - om profilbilder skal vises med knappen

max_rows - maks antall rader med profilbilder

width - bredde på knappen.

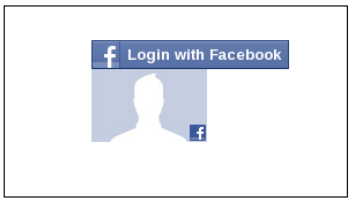
perms - forespørsel om utvidede tillatelser

size - strørrelse på knappen, large/normal

versjon - versjon av knappen, ny/original

onlogin - eksekvere koden ved innloggingen

Høyden på knappen justeres dynamisk etter om venners profilbilder skal vises som en del av innloggingsknappen, og eventuelt hvor mange venners profilbilder som skal vises. Slik kan en innloggingsknapp se ut:



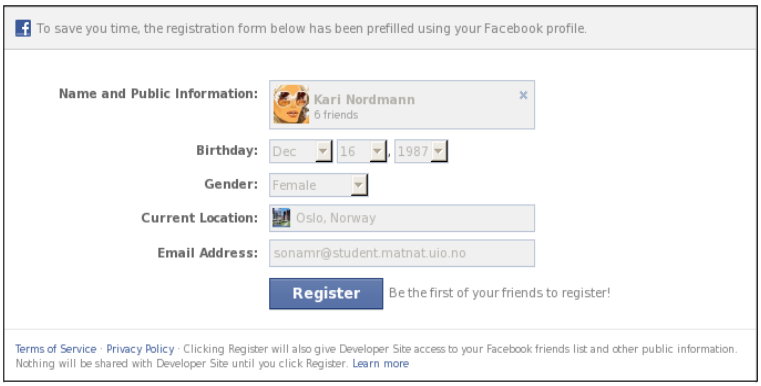
Figur 6.1: Facebook JavaScript SDK: Login-Button.

Ettersom at utvikleren kun trenger å implementere elementet, fb:login button, så kalles en autentisering og autorisasjon med JavaScript SDKet for en enkel innlogging (simple login).

Dersom utvikleren av en tredjepartsapplikasjon ønsker å bruke data om brukere som ikke er tilgengelig via graf-APIet, kan hun med JavaScript SDKet implementere en *Registration plugin*. En registrerings-plugin tillater brukere å enkelt registrere seg i applikasjoner [53]. Registrerings-plugin'en

er en enkel iframe[2] i applikasjonssiden og er uavhengig av en Facebook-brukerkonto, i motsetning til innloggingsknappen. På denne måten støtter registrerings-plugins også brukere uten Facebook-kontoer.

For å bruke registrerings-plugins, trenger utvikleren å inkludere elementet, *fb:registration*. Når brukere laster inn applikasjonssiden i en nettleser, vil Facebook presentere et registreringsskjema for brukeren. Dersom brukeren har en Facebook-konto og allerede er logget inn i Facebook, vil skjemaet være ferdigutfylt med brukerens informasjon, i de feltene som inneholder informasjon som allerede finnes i Facebook, som *navn* og *e-post adresse*. Dersom brukeren ikke er har en brukerkonto eller ikke er logget inn, så vil skjemaet være tomt, og klart til at brukeren kan oppgi sin informasjon. Informasjon som ikke finnes i graf-APIet og som utvikleren ønsker om brukerne, angis i element-attributtet *fields*. Plugin'en åpner for tilpasning ettersom at utvikleren kan opprette egendefinerte felter, eksempelvis *favoritt bil* eller *SIP adresse*. Slike egendefinerte felter fylles ikke automatisk ut, selv om brukeren allerede er logget inn i Facebook. Plugin'en er utstyrt med en *Registrer*-knapp nederst i skjemaet. Når informasjonen i skjemaet er fylt og knappen er trykket på, så vil autentiseringen og autorisasjonen utføres av JavaScript SDKet, på samme måte som med innloggingsknappen.



Figur 6.2: Facebook JavaScript SDK: Registrerings-plugin.

Se [60] for en fullstendig oversikt over JavaScript SDKet.

6.3 Social Plugins

Innloggingsknappen og registrerings-plugin'en i seksjonen ovenfor er eksempler på *Social Plugins*. Facebooks sosiale plugins er den enkleste må-

[2]En iframe (inline frame) er et HTML-dokument plassert inn i et annet HTML dokument. Iframe-elementet brukes ofte til å sette inn innhold fra en annen kilde, inn i en webside.

ten å integrere en tredjepartsnettside eller applikasjonsside med Facebook-plattformen. Det finnes en rekke forskjellige sosiale plugins og tillater brukere å se hva deres venner liker, har kommentert på og delt på andre sider.

Alle sosiale plugins er ekstensjoner av Facebook, og er spesielt designet for at data om brukeren ikke skal være tilgjengelig for siden som implementerer plugin'en, selv om brukeren benytter plugin'en [49]. Med andre ord, så åpner Facebook for at brukere skal kunne benytte seg av Facebook-plugins på eksterne nettsider og applikasjonssider, uten at sidene har innsyn i informasjon om brukeren. Innlogging og registrerings-plugin'ene er unntak, da de trenger informasjon om brukeren for å utføre autentisering. Plugins har vist seg å være populært både for utviklere av tredjepartssider og Facebook-brukere, ettersom over 50 000 publiseringer ble gjort via sosiale plugins, èn måned etter at plugin'ene ble introdusert [41].

Det finnes to måter å implementere sosiale plugins på, enten med *Extensible Facebook Markup Language (XFBML)* eller med en iframe. XFBML er utviklet av Facebook-utviklere og er essensielt XML begrenset til å ikke akseptere enkelte XML-elementer og å akseptere noen nye elementer definert av Facebook, slik at XML er tilpasset Facebook. XFBML-varianten er mer allsidig, men krever at utvikleren bruker JavaScript SDKet. XFBML endrer dynamisk på høyden til plugin'en, og tilbyr også utvikleren å lytte på handlinger slik at hun i sanntid får beskjed når en bruker benytter plugin'en, gjennom JavaScript SDKet. Dersom plugin'en er implementert med en iframe, blir den et Facebook-uavhengig element i siden.

Det finnes forskjellige type plugins, og vi skal i de følgende underseksjonene se på noen av dem.

6.3.1 Like Button

Den mest populære plugin'en har vist seg å være, *Like Button* plugin, som lar Facebook-brukere uttrykke preferanser eller «like» innhold, i bare ett klikk. Når brukeren trykker på en Like-knappen som er implementert i en ekstern nettside, vil en publisering bli gjort tilbake i nyhetsstrømmen på hjemsiden til brukerens Facebook-profil og i brukerens brukerprofil, med en link til den eksterne nettsiden som implementerer pluggen. Dersom plugin'en er implementert i en tredjepartsapplikasjon, så vil det på samme måte bli gjort en publisering til brukerens brukerprofil og nyhetsstrøm, med en link til applikasjonssiden. Kodesnutten for en Like-knapp i en ekstern side med XFBML er som vist nedenfor, hvor script-elementet tillater JavaScript og XFBML-kode, og «www.din_nettdide.com» angir url-en til den eksterne siden hvor pluggen skal implementeres. Dersom plugin'en skal implementeres i en tredjepartsapplikasjon, trenger href-attributtet ikke å oppgis:

```
<script src="http://connect.facebook.net/en_US/all.js#xfbml=1"></script>
<fb:like href="www.din_nettside.com"></fb:like>
```

Det finnes en rekke attributter med knappen:

layout - stil på knappen. Standard = sosial tekst til høyre for knappen og profilbilde til venner som har brukt knappen, under. Button_count = viser antall som har trukket på knappen til høyre for knappen.

show_faces - spesifiserer om profilbilder skal vises med knappen

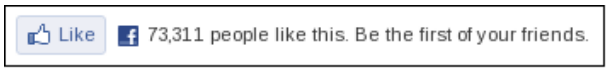
width - bredden på knappen

action - verbet som skal stå på knappen, Recommend eller Like.

font - font på knappen.

colorscheme - fargetema på knappen, light eller dark.

En like-knapp kan se slik ut:



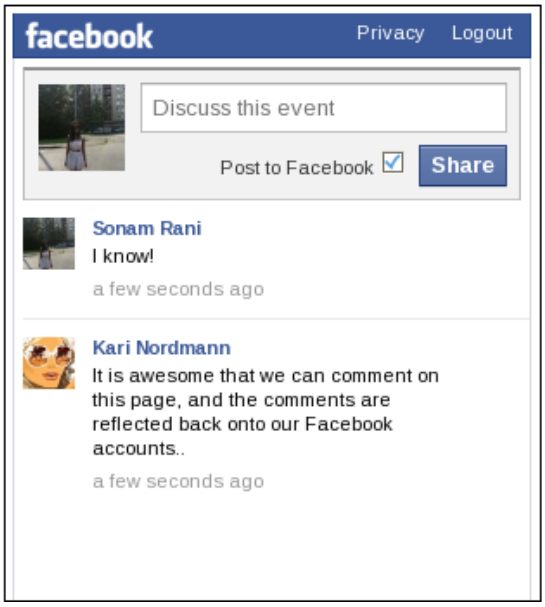
Figur 6.3: Facebook sosiale plugins: Like-knapp.

6.3.2 Live Stream

Live Stream-plugin'en lar brukere besøke utvikleres applikasjoner eller eksterne nettsteder og dele aktiviteter eller kommentarer, i sanntid. Plugin'en har som formål å la brukere interagere under sanntidshendelser, som direkteavspilling (live streaming) av konserter, spill, video, taler, konferanser, og lignende. Sanntidsstrømmings-plugin'en krever at brukeren har en applikasjons-ID som er knyttet til nettsiden eller applikasjonen som skal implementere plugin'en. Plugin'en kan implementeres med følgende XFBML-kode, der «din_appID» angir applikasjons-IDen for nettstedet:

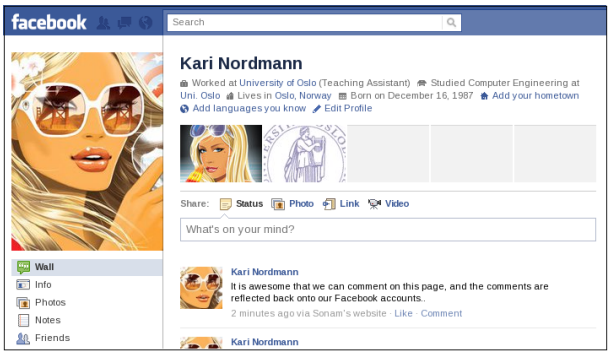
```
<fb:live-stream event_app_id="din_appID"></fb:live-stream>
```

I tillegg til høyde- og vidde-attributtet har pluggen et «xid»-attributtet. Dette er til å skille mellom pluggene dersom flere sanntidsstrømmings plugins er implementert i en og samme side. Dette er et eksempel på en sanntidsstrømmings plugin:



Figur 6.4: Facebook sosiale plugins: Sanntidsstrømmings plugin.

I denne figuren ser vi hvordan en kommentar i en live stream-boks på en ekstern nettside reflekteres tilbake til brukeren Facebook-profil:



Figur 6.5: Facebook sosiale plugins: Sanntidsmeldinger.

Innloggingsknappen, registrerings-plugin'en, like-knappen og sanntidsstrømmings plugin'en er noen eksempler på Facebook sine sosiale plugins. I tillegg til disse finnes blant annet plugins som *Recommendations* som tillater brukere å anbefale innhold i sider, *Comments* som lar brukere kommentere på innholdsbiter i sider og *Facepile* som viser profilbildet til venner av brukeren som er liker eller er logget inn i en side. For mer om XFBML, se [61]. For mer om sosiale plugger, se [56, 49].

6.4 Andre SDKer og APIer

Facebook tilbyr en rekke alternative SDKer, som:

- PHP SDK
- iOS SDK
- Android SDK

PHP programvareutviklingsverktøyet har også åpen kildekode og er tilgjengelig for utviklere, og brukes i like stor grad som JavaScript SDKet. En detaljert beskrivelse av verktøyet finnes i [14]. Over 250 millioner av Facebook sine brukere, logger månedlig inn i nettsiden via Facebook for mobiltelefoni[59]. Facebook iOS SDKet gir utviklere muligheten til å integrere Facebook i iOS-applikasjoner, som i *iPhone*, *iPad* og *iPod touch* [55]. Tilsvarende tillater Android SDKet utviklere å integrere Facebook i Android-applikasjoner [12, 13].

Facebook har utviklet *Open Graph Protocol* for å la eiere av eksterne nettsider integrere deres sider inn i Facebook sin sosiale graf, slik at de representeres som objekter i Facebook sitt graf-API. Til forskjell fra Facebook sine sosiale plugger, er Open Graph protokollen ment for nettsider som representerer profiler fra den virkelige verden, som: filmer, sportslag, kjendiser, resturanter, butikker og lignende. Eiere kan gjøre sin nettside ekvivalent med en Facebook-side, ved å implementere Open Graph Protocol. Nettsiden vil finnes blant alle andre Facebook-sider, og vil også vises i Facebook søkefeltet, som om nettsiden skulle vært en opprinnelig Facebookside. Mer om Open Graph protokollen finnes i [62].

Kapittel 7

Facebook-applikasjonen: VoIPBook

7.1 Om integreringen

I de foregående kapitlene har vi sett på hva Facebook tilbyr av APIer og SDKer til utviklere, for å tillate integrasjon med Facebook-plattformen. Videre skal vi se på hvordan disse verktøyene kan brukes til å integrere VoIP-tjenester i Facebook-plattformen. Hovedformålet med integrasjonen er å få en økt nettverkseffekt for VoIP-tjenester, og som vi har sett så kan en økt nettverkseffekt finnes i å tilby et kontaktoppslagsverk av SIP-adresser, og å la brukere ringe med VoIP via oppslagsverket. I de neste kapitlene og seksjonene skal jeg demonstrere en Facebook-applikasjon, *VoIPBook*, som tilbyr slike VoIP-tjenester.

VoIPBook er en Facebook-applikasjon, altså en applikasjon som oppholder seg i Facebook-domenet. Alternativet til en slik applikasjon, er å ha en tredjeparts nettside som inneholder VoIP-tjenesten, og som integrerer med Facebook-plattform eksempelvis via sosiale plugins eller ved å implementere Open Graph Protocol.

Sosiale plugins tillater ikke utvikleren innsyn i informasjon om brukeren av plugin'en, slik at utvikleren vil dermed ikke være i stand til å presentere brukere med Facebook-data om brukeren selv eller brukerens venner. I en VoIP-løsning med sosiale plugins må utvikleren få informasjon om brukeren og brukerens venner fra en annen kilde for å presentere det for applikasjonsbrukere. Plugin'ene vil dermed kunne brukes til å publisere aktiviterer fra tredjepartssiden som tilbyr VoIP-tjenesten, i Facebook.

En tredjepartsnettside som implementerer Facebook sitt Open Graph Protocol tillater nettsiden å bli en integrert del av Facebook. Dette innebærer at nettsiden vil bli en del av Facebook sin sosiale graf, og representeres som et objekt i grafen. Dersom vi velger en slik løsning, må VoIP-tjenestene være fullstendig implementert i tredjepartsnettsiden. Implementasjon av

Open Graph Protocol ville tilgjengeliggjort en allerede eksisterende VoIP-tjeneste via Facebook-plattformen. Imidlertid så vil utvikleren ikke være i stand til å hente og fremvise Facebook-data i tredjepartssiden. I en slik løsning blir altså siden en del av Facebook.

Jeg har valgt at VoIPBook skal være en Facebook-applikasjon, for at VoIP-tjenestene som applikasjonen tilbyr skal «virke» som en integrert del av Facebook, i stedet for at tjenesten skal virke som en del av en annen side. Med en slik løsning vil utvikleren være i stand til å presentere Facebook-informasjon om brukeren og hans venner, og i tillegg tillate brukeren å oppgi informasjon til applikasjonen. Som en applikasjon som befinner seg i Facebook-domenet, vil tjenesten også tilgjengeliggjøres mer for brukere.

For å bygge opp en applikasjon i Facebook-domenet, så må man opprette et Facebook-objekt med PHP SDKet. Ettersom at PHP SDKet uansett må lastes inn i siden, har jeg valgt å benytte dette SDKet til videre forespørsler i applikasjonen. Alternativt kunne vi brukt JavaScript SDKet til å utføre forespørsler i VoIP-tjenesteapplikasjonen. I VoIPBook hentes informasjon fra Facebook-grafen hovedsakelig med HTTP forespørsler på graf-APIet, da jeg i applikasjonen ikke har behov for å utføre komplekse spørringer om Facebook-data, slik som FQL tillater. Den viktigste informasjonen vi ønsker å opprettholde i applikasjonen, nemlig brukeres SIP-adresser, er data som ikke hentes fra Facebook-grafen, men som oppgis av applikasjonsbrukerne.

7.2 Applikasjonsoppsett

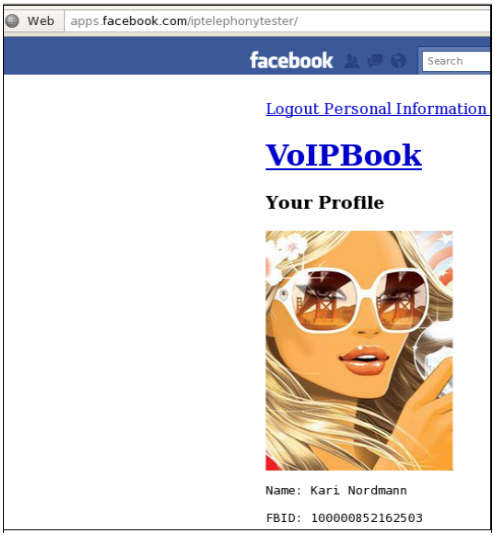
Applikasjoner i Facebook-domenet er web-applikasjoner som lastes inn i sammenheng med Facebook. Alle slike applikasjoner lastes inn i en *Canvas Page* i Facebook. En slik Canvas-side er et blankt lerret innen Facebook, som applikasjoner kan kjøre på. Alle Canvas-sider som tilhører applikasjoner i Facebook, har samme prefiks, *http://apps.facebook.com/*. For å entydig identifisere en applikasjon, må man angi et navn til Canvas-siden. Adressen som brukere må angi i en nettleser for å omdirigeres til applikasjonens Canvas-side, er Canvas-prefiksen etterfulgt av navnet til Canvas-siden. Canvas-siden til applikasjonen jeg har utviklet heter *IP Telephony Tester*, slik at adressen til applikasjonen vil dermed bli
http://apps.facebook.com/iptelephonytester. Selveste applikasjonsnavnet er VoIPBook.

Man kan bygge ut Canvas-sider med ethvert programmeringsspråk og verktøy som støtter web-programmering, som blant annet HTML, CSS, JavaScript og PHP. For å fylle ut applikasjonen på en slik måte, må man angi hvor koden til applikasjonen finnes. Denne kilden kalles en *Canvas URL*. For å utvikle applikasjonen min benytter jeg domeneområdet jeg er blitt tildelt fra Universitet i Oslo, som er *http://sonamr.at.ifi.uio.no/*. Filene til Canvas-siden har blitt plassert i undermappen /php/Master, som gjør at

Canvas-URL'en til applikasjonen blir *http://sonamr.at.ifi.uio.no/php/Master/*. Canvas URL'ens innhold lastes inn i en iframe på Canvas-siden til applikasjonen, slik at innholdet er uavhengig av Facebook.

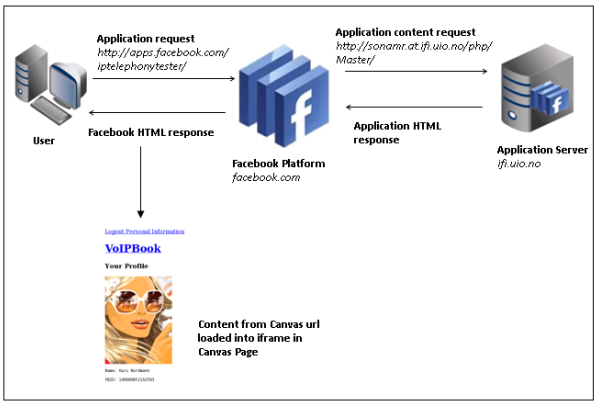
For å opprette applikasjonen, må den registreres i et skjema i Facebook-domenet. Her angis også blant annet navnet til applikasjonen og Canvas-siden, og også Canvas-URL'en. Applikasjonen vil bli tildelt en unik *Applikasjons-ID*. Utvikleren tildeles også en *Applikasjonsnøkkel* og *Applikasjonshemmelighet* som kan brukes til forespørseler mot graf-APIet til Facebook.

Jeg har opprettet en fiktiv Facebook-brukerkonto under navnet *Kari Nordmann*, som vil stå som utvikler av applikasjonen, og som jeg bruker til testingsformål i oppgaven. Canvas-siden til VoIPBook-applikasjonen ser slik ut for Kari Nordmann, etter å ha hentet ut profilbildet, Facebook-ID og navn fra graf-APIet:



Figur 7.1: VoIPBook Canvas-side.

En Facebook-applikasjon oppfører seg som en typisk webside da den lastes inn fra en webserver, og serverer innhold til HTTP forespørsler. Forskjellen fra en typisk webside er at i Facebook-applikasjoner vil brukerforespørsler gå via Facebook-plattformen til applikasjonen, i stedet for at applikasjonssiden mottar forespørsler direkte fra brukere. Adressen til applikasjonens Canvas-side er en *frontadresse* i Facebook-domenet. I VoIPBook er denne adressen *http://apps.facebook.com/iptelephonytester*. Brukeren aksesserer denne frontadressen for å besøke applikasjonen, mens Facebook videre kontakter *originaladressen* på egenhånd. Denne originaladressen er Canvas-URL'en, som i VoIPBook-tilfellet er *http://sonamr.at.ifi.uio.no/php/Master/* [3]:



Figur 7.2: Facebook-applikasjoners arkitektur.

Altså, brukere navigerer seg til frontadressen i nettleseren deres for å besøke applikasjonen. Dette vil føre til at Facebook henter innholdet til Canvas-siden, ved en forespørsel til Canvas-URL'en mot en av UiO sine servere, der applikasjonskoden er lagret. Responsen vil i sin tur returneres til Facebook webservere, og innholdet vil bli lastet inn i en iframe på Canvas-siden for brukeren.

Etter at applikasjonen er opprettet, lager jeg filene med koden som ønskes i applikasjonen, under domeneområdet jeg er tildelt på Universitetet i Oslo. Applikasjonen starter i filen, *index.php*. En *index-side* er siden som vises når en besøker toppnivået av en webside. Dersom man laster inn *http://sonamr.at.ifi.uio.no/php/Master/* i en nettleser, så vil nettleseren som en standard vise innholdet i index-filen, da ingen annen fil er angitt. Nettadressen *http://sonamr.at.ifi.uio.no/php/Master/* vil derfor være identisk med *http://sonamr.at.ifi.uio.no/php/Master/index.php*.

I index-filen opprettes først og fremst et *Facebook-objekt*. Dette objektet representerer web-applikasjonen som er opprettet i Facebook-domenet, og behøves for at utviklere for å bygge opp applikasjonen, utføre autentisering og autorisasjon, og å sette og hente ut informasjon om en sesjon. Objektet tar applikasjons-ID'en og applikasjonshemmeligheten som argumenter til konstruktøren. Det finnes i tillegg en rekke opsjonelle parametere:

```
$facebook = new Facebook(array(
  'appId'  => '110822422284818',
  'secret' => 'c9e2426a4ca370a3eb44xxxxxxxxxxx',
  'cookie' => true,
));

$facebook->setBaseDomain('http://sonamr.at.ifi.uio.no/php/Master/');
$session = $facebook->getSession();
```

Facebook-klassen med de tilhørende klassevariabeler, objektvariabler og metoder finnes i filen *facebook.php*, som tilbys av Facebook-utviklerne som en del av PHP SDKet. Tredjepartsutviklere får dermed tildelt koden til klassen, men må implementere objektet på egenhånd. For en fullstendig implementasjon av Facebook-objektet til VoIPBook-applikasjonen, se *index.php* i Vedlegg A.

7.3 Autentisering og autorisasjon

7.3.1 OAuth 2.0

Facebook tilbyr en rekke måter å la brukere av en applikasjon *autentisere* seg på. Som vi så i forrige kapittel kan utviklere logge inn brukere og utføre autentisering og autorisasjon i applikasjonene deres på en enkel måte, med JavaScript SDKet. En alternativ måte er ved bruk av *OAuth 2.0*-protokollen. OAuth 2.0 er en lettere versjon av den standarde protokollen *OAuth (Open Authorization)*. Som en tredjepartsapplikasjon, trenger man aksess til ressurser som opprettholdes av Facebook-servere. Disse ressursene er som regel beskyttet og krever autentisering ved bruk av ressurseierens legitimasjon,

som typisk er et brukernavn og passord. I den tradisjonelle klient-server autentiseringsmodellen, så aksesserer klienten en beskyttet ressurs på serversiden, ved å autentisere seg med serveren ved å bruke ressurseierens legitimasjon. For å tilby tredjepartsapplikasjoner aksess til beskyttede ressurser kan altså ressurseieren dele sin legitimasjon med tredjeparten. Dette kan skape en rekke problemer og begrensninger. Tredjeparten må typisk lagre ressurseierens legitimasjon for senere bruk, slik at det hender at passordet lagres i klartekst. Tredjepartsapplikasjoner innvilges unødvendig bred aksess til de beskyttede ressursene da ressurseieren ikke har noen mulighet til å begrense aksessen til et subsett av ressursene eller begrense tidsintervall for aksessen. Resurseieren kan heller ikke frata aksess fra en individuell tredjepart uten å frata aksessen fra alle tredjeparter, og må i såfall gjøre det ved å endre passordet sitt [20].

OAuth tar hensyn til problemene som kan oppstå med tradisjonell klient-server autentisering. I OAuth, vil klienten forespørre aksess til ressursene kontrollert av ressurseieren og som opprettholdes av ressursserveren, tildeles et sett med legitimasjon som er *forskjellig* fra ressurseierens legitimasjon [20]. I stedet for å bruke ressurseierens legitimasjon til å aksessere beskyttede ressurser, mottar klienten et *access token*. Et slikt aksesslodd er en string som angir et spesifikt skop, varighet og andre attributter. Aksesslodd tildeles brukere av tredjepartstjenester av en autoriseringsserver med godkjenning av ressurseieren. Brukeren benytter så dette aksessloddet til å aksessere de beskyttede ressursene på ressursserveren.

Tilsvarende i vår applikasjon, trenger vi at Facebook (ressurseier) sine webservere innvilger aksess til kontaktoppslagstjenesten via VoIPBook (tredjepart) hvor kontaktinformasjonen er lagret i en av Facebooks datadelingstjenester (ressursserver), uten at Facebook webservere deler sin legitimasjon med kontaktoppslagstjenesten. I stedet må brukere som forespør kontaktoppslagstjenesten ved bruk av VoIPBook (klienter) autentisere seg direkte med en autentiseringstjeneste (autentiseringsserver) som er betrodd av datadelingstjenesten, og som tildeler kontaktoppslagstjenesten legitimasjon (aksesslodd) for klienten. For mer om OAuth, se [20].

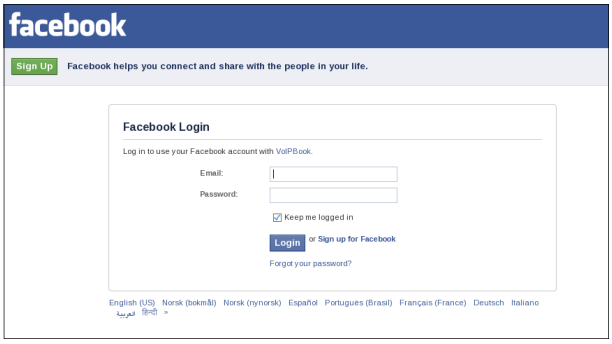
I VoIPBook benytter jeg autentisering og autorisasjon med OAuth 2.0. Det er tre steg til å fullføre en slik prosess. Det første steget er *brukerautentisering* som skal forsikre at brukeren er den han påstår å være. Det andre steget er *applikasjonsautorisasjon* som skal sørge for at brukeren vet nøyaktig hvilken data og kapasiteter han innvilger applikasjonen. Det siste steget er *applikasjonsautentisering*, som skal forsikre brukeren om at han gir sin informasjon til applikasjonen han autoriserer, og ingen annen. Først etter at disse tre stegene er fullført, vil applikasjonseieren bli tildelt et aksesslodd, som kan brukes til å hente ut informasjon og utføre handlinger på vegne av brukeren.

Bruker autentisering og applikasjons autorisasjon utføres i ett, ved å omdirigere brukeren til Facebook sin OAuth Dialog. I omdirigeringen må

applikasjons-ID'en angis i *client_id* parameteren, samt URL'en som brukerens nettleser skal omdirigeres tilbake til etter at bruker autentiseringen og applikasjonsautorisasjonen er utført. Denne URL'en angis i parameteren *redirect_uri* og må være i samme domenet som du oppgir i oppsettet av applikasjonen. I applikasjonen setter jeg omdirigeringen til å være index-filen:

```
https://www.facebook.com/dialog/oauth?
     client_id=110822422284818
     &redirect_uri=http://sonamr.at.ifi.uio.no/php/Master/index.php
```

Dersom brukeren allerede er logget inn i Facebook, så vil innloggings-cookie'en valideres og autentiseringen av brukeren blir gjort. Dersom brukeren ikke er innlogget, åpnes et innloggingsvindu hvor brukeren kan oppgi legitimasjon og autentiseres:



Figur 7.3: VoIPBook: Autoriseringsgrensesnitt.

Etter at brukeren er autentisert, så vil autorisasjonsprosessen begynne. Når en bruker autoriserer en applikasjon så har applikasjonen som standard aksess til all generell informasjon i en brukerprofil. For å aksessere annen informasjon må utvidede tillatelser gis under autoriseringen. Under autentiseringsprosessen av applikasjonen, vil brukeren bli presentert med et brukergrensesnitt, hvor brukeren kan autorisere applikasjons aksesser. I VoIPBook, forespør jeg følgende utvidede tillatelser:

manage_pages - For å gi applikasjoen tillatelse om å innhente en aksess-token for sider brukere eventuelt administrerer.

publish_stream - For å tillate applikasjonen å publisere innhold i brukerens strøm.

sms - For å tillate applikasjonen å sende/respondere på meldinger til/fra brukeren via tekst-meldinger.

offline_access - For å tillate applikasjonen å utføre autorisasjons-forespørseler på vegne av brukeren til enhver tid, som også gjør at aksess-token som brukeren utleverer har lengre levetid.

user_about_me - For å få aksess til *About me*-seksjonen på brukerens profil, hvor blant annet kontaktinformasjon oppgis.

friends_about_me - For å til aksess til *About me*-seksjonen til venner av brukeren.

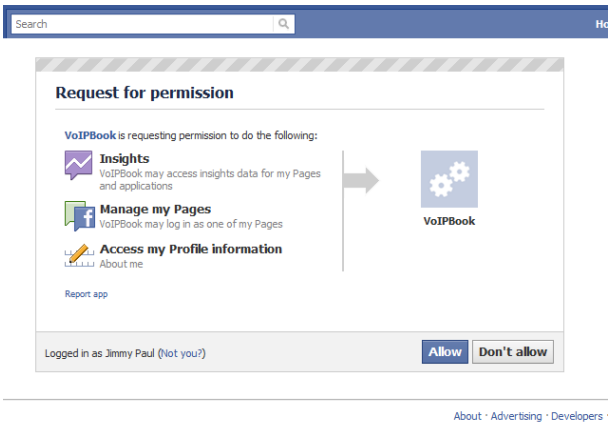
user_online_presence - For å få tillatelse til innsyn om brukeren er pålogget for øyeblikket.

email - For å få aksess til brukerens epost-adresse, dersom den er oppgitt.

De utvidede tillatelsene som applikasjonen ønsker autorisering til, angis sammen med omdirigeringen til OAuth Dialogen, under parameteren *scope*:

```
https://www.facebook.com/dialog/oauth?
     client_id=110822422284818
     &redirect_uri=http://sonamr.at.ifi.uio.no/php/Master/index.php
     &scope=manage_pages,publish_streams,sms,offline_access,
     user_about_me,friends_about_me,user_online_presence,email
```

Autoriseringsgrenesnittet vil lanseres rett etter bruker autentiseringen, og ser slik ut:



Figur 7.4: Autoriserings-grensesnitt.

Dersom brukeren trykker *Don't Allow* så vil applikasjonen ikke autoriseres, og omdirigere til URL'en oppgitt i redirect_uri-parameteren med en feilmelding. Dersom brukeren trykker *Allow*, så er applikasjonen autorisert, og nettleseren vil omdirigere til URL'en som ble oppgitt i redirect_uri-parameteren med en *Autorisasjonskode* i parameteren *code*.

Autorisasjonskoden som returneres etter en vellykket autorisasjon, brukes videre til å autentisere applikasjonen, og oppnå et aksesslodd. For å autentisere applikasjonen, må brukeren omdirigeres til graf-APIets *token endpoint*. Også i denne omdirigeringen må det oppgis applikasjons-ID'en og en omdirigerings-URL som angir hvilken side det skal omdirigeres tilbake til, samt noen ytterligere parametre. Nå må også applikasjonshemmeligheten angis i parameteren *client_secret*, og autorisasjonskoden som returnertes i kallet over, må angis i parameteren *code*:

```
$code = $_REQUEST['code'];
https://graph.facebook.com/oauth/access_token?
     client_id=110822422284818
     &redirect_uri=http://sonamr.at.ifi.uio.no/php/Master/index.php
     &client_secret=c9e2426a4ca370a3eb44xxxxxxxxxx
     &code=$code
```

Dersom det oppstår problemer med autentiseringen av applikasjonen, så vil autorisasjonsserveren returnere en feilmelding, og ingen aksesslodd vil

bli utstedt. Dersom applikasjonen er autentisert og autorisasjonskoden er gyldig, vil autorisasjonsserveren returnere et aksesslodd. Dette aksessloddet finnes i parameteren *access_token*. Sammen med aksessloddet returneres også loddets levetid, i parameteren *expires*. Når aksessloddet utløper må stegene ovenfor utføres på nytt. Dersom offline_access tillatelsen er forespurt og innvilget, så vil utløpstiden for aksessloddet være langt, og prosessen trenger ikke å gjentas så ofte. Etter å ha oppnådd et aksesslodd, kan utvikleren av applikasjonen utføre handlinger på vegne av brukeren innenfor rammene som tillatelsene danner.

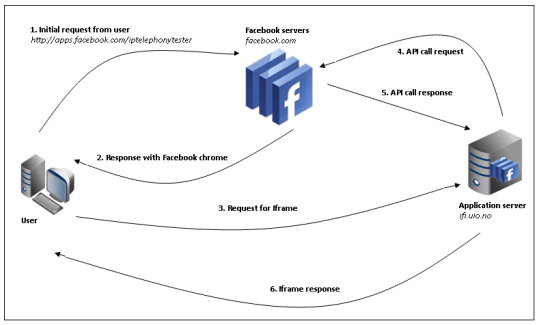
For at applikasjon skal gi en god brukeropplevelse, er det viktig at utseende på applikasjonen er pent og at brukerne blir fremvist med personlig informasjon og informasjon om venner på en oversiktelig måte. I neste seksjon skal vi se på hvordan VoIPBook henter ut slik data fra graf-APIet ved å benytte applikasjonsbrukernes oppnådde aksesslodd. Vi skal også se på hvordan applikasjonsbrukere kan oppgi SIP-adresser til kontaktoppslagstjenesten, og hvordan VoIPBook presenterer og holder på denne informasjonen.

7.4 Dataopphenting

Applikasjoner i Facebook er såkalte *Iframe Canvas*-applikasjoner, da de bruker iframe'ene som datafremvisningsmekanisme. Data som fremvises i applikasjoner kan være informasjon som applikasjonen henter fra Facebookgrafen, og også informasjon som er hentet fra andre kilder.

Når en bruker laster inn en side i applikasjonen og skal fremvises med applikasjonsdata, så sendes en brukerforespørsel fra brukerens nettleser til Facebook sine webservere, slik som i figur 7.5. Facebook-servere vil respondere med sitt Facebook *krom*[1], som tilpasser sidene iframe'ene skal fremvise datainnholdet på, ettersom iframe-applikasjonene er Facebookuavhengige. Videre så vil brukerens netteleser forespørre applikasjonsserveren om iframe med ønsket innhold. Applikasjonsserveren vil videre gjøre et API-kall mot Facebook-servere, for å hente ut informasjon fra Facebookgrafen, som eventuelt skal være med i sidefremvisingen. I sin tur vil Facebookserverne sende en API-respons til applikasjonsserveren, med forespurt innhold. Denne responsen vil applikasjonsserveren gjengi som HTML-kode i en iframe til brukerens nettleser, før innholdet i iframe'en fremvises for brukeren på applikasjonens Canvas-side. Iframe-applikasjoner bruker dermed ikke Facebook som mellomledd mellom applikasjonsserveren og brukeren. Figur 7.5 illustrerer hvordan applikasjonsdata hentes og fremvises i VoIPBook:

[1] Et krom (chrome) er grensene til et nettleser-vindu, inkludert vinduets rammer, menyer, verktøylinje, og rullefelt.



Figur 7.5: IFrame Canvas Applikasjons: Informasjonsflyt.

7.4.1 Personlig informasjon

Ettersom index-siden ikke bare utfører det obligatoriske rundt oppsettet av applikasjonen men også er den første siden som fremvises til applikasjonsbrukere, så har jeg latt siden inneholde litt informasjon om brukeren selv, og lar den oppleves som en samleside.

For å hente ut informasjon om brukeren, gjør jeg til å begynne med et API-kall med metoden *api()* med argumentet *me* med PHP SDKet, for å hente informasjon om brukeren selv. Metoden returnerer et JSON objekt med generell og utvidet informasjon om brukeren av applikasjonen. I index-siden skriver kun jeg ut den mest essensielle informasjonen om brukeren, som: navn, Facebook-ID, Fødselsdato og e-postadresse. Jeg forespør også objektet som inneholder bildet til applikasjonsbrukeren med en HTTP forespørsel, på grunnlag av hans Facebook-ID. Ettersom at profilbildet går under generell informasjon, behøves ikke aksessloddet til å utføre spørringen:

```
$me = $facebook->api('/me');

echo 'Name: '; print_r($me['name']);
echo 'FBID: '; print_r($me['id']);
echo 'DoB: '; print_r($me['birthday']);
echo 'Email: '; print_r($me['email']);

$fbid = $me['id'];
print '<img src="https://graph.facebook.com/' .$fbid . '/picture?type=large"/>';
```

Ettersom VoIPBook blant annet skal være en kontaktoppslagstjeneste, bør brukeren også ha muligheten til å se en fullstendig oversikt over personlig data som andre applikasjonsbrukere kan ha innsyn i. Dette er all generell og utvidet informasjon om brukeren, som applikasjonen har fått aksess til. Dataene fremvises ved å trykke på en egen link med teksten «Personal Information» som er plassert ved inn/utloggings linken øverst på index-siden. Når brukeren trykker på linken, vil brukeren bli omdirigert til siden *me.php*, hvor resultatet av grapf-spørringen fremvises.

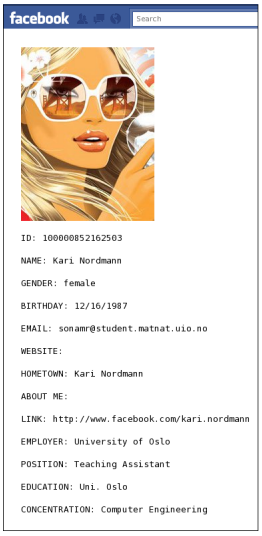
Spørringen gjøres ved å igjen gjøre et api-kall på *me*-objektet, og filtrere feltene på informasjonen vi ønsker:

```
$my_info = $facebook->api('/me?fields=id,name,gender,
birthday,email,location,about,link,work,education');
...
```

Den returnerte JSON-responsen parseres riktig, og sendes som URL-parametere til me.php-filen. Parameterne hentes ut i me.php, og fremvises for applikasjonsbrukeren:

```
...
echo "<a href='http://sonamr.at.ifi.uio.no/php/Master/me.php?id=$user_id
&name=$user_name
&gender=$gender
&birthday=$birthday
&email=$email
&website=$website
&hometown=$home
&abt=$abt
&link=$link
&employer=$employer
&position=$position
&school=$schoolname
&field=$field  '> Personal Information </a>";
```

Når brukeren trykker på linken ovenfor vil informasjonen fremvises på følgende måte:



Figur 7.6: VoIPBook: Side med fullstendig oversikt over personlig informasjon tilgjengelig i applikasjonen..

Se *index.php* for parsering av responsen. Filen finnes under Vedlegg A.

7.4.2 Informasjon om venner

Som en kontaktoppslagstjeneste så trenger applikasjonsbrukere hovedsakelig å se informasjon om deres venner.

For å hente ut informasjon om brukerens venner, benytter jeg også her api()-metoden fra PHP SDKet. Spørringen som utføres er */me/friends?*, da

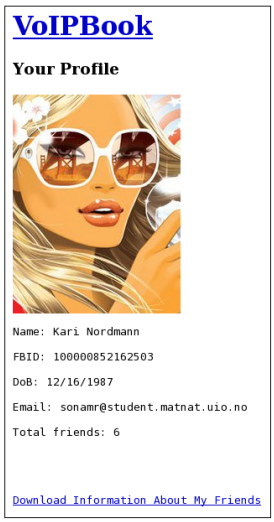
vi ønsker å spørre om brukerens me-objektet hvor forbindelsen vi ønsker er brukerens venner. Spørringen inkluderer kun feltene *id*, *name* og *email*. Merk at e-post informasjon refererer til den primære e-postadressen Facebook-brukere oppgir som sin kontaktinformasjon og anses som sensitiv informasjon. Av sikkerhetsmessige grunner betyr dette at selv om brukeren selv har tillatt applikasjonen utvidet aksess til informasjon om brukerens venner, har applikasjonen *ikke* tilgang til epostadressen til brukerens venner. Så for at feltet skal ha en verdi, så må tilgangen eksplisitt innvilges av brukerens venner.

Objektet som returneres av api-kallet, parseres og skrives til fil. Hvert filnavn består av teksten *FacebookFriendFile*, konkatenert med applikasjonbrukerens Facebook-ID, slik at hver fil er identifiserbar for en bruker. Et eksempel på en slik fil til bruker med Facebook-ID «100000852162503», er «FacebookFriendFile-100000852162503.txt». Til filen skrives brukerens venners Facebook-IDer og navn. Ettersom de fleste epost-felter står tomme, unnlates de å skrives til fil. Dersom applikasjonsbrukeren ikke er en ny bruker av applikasjonen og dermed har en slik fil fra før av, så overskrives foregående fil hver gang brukeren logger inn i applikasjonen, da brukeren kan ha fått nye relasjoner på Facebook i mellomtiden. For hver nye bruker som besøker applikasjonen, så vil det opprettes en slik fil. Filene lagres under mitt webområde i samme mappe som index-filen. Brukeren kan også nedlaste filen fra siden, dersom han ønsker å flette informasjon om sine Facebook-venner til andre adressebøker.

Samtidig som brukerens venner skrives til fil, opprettholder jeg en teller, *friendCounter*, som holder på antall venner brukeren har. Denne verdien skrives også ut på index-siden:

```
$friends = $facebook->api('/me/friends?fields=id,name,email');

$friendFile = "FacebookFriendFile" ."-" . $me['id'] . ".txt";
$fileHandle = fopen($friendFile, 'w') or die("can't open file");

$friendCounter = 0;
for ($i=0; $i<= filesize($friendFile); $i++) {
 ....

  $friendCounter++;

  $data_string = $f['id'] . " : " . $f['name'] . ";";
  fwrite($fileHandle, $data_string);
  fwrite($fileHandle, "\n");

   }
fclose($fileHandle);

echo 'Total friends: ' . $friendCounter;

<a href="http://sonamr.at.ifi.uio.no/php/Master/<?=$friendFile?>"
>Download Information About Your Friends</a>
```

Et eksempel på en slik profil-side kan være som dette:



Figur 7.7: VoIPBook: Hovedside med antall av applikasjonsbrukerens relasjoner, og mulighet for å nedlaste fil om venner.

Formålet med å lage en fil per applikasjonsbruker, er at vi ønsker å holde på informasjon om applikasjonsbrukeres venner, for videre prosessering i tjenesten. Å opprette en slik fil for hver bruker av applikasjonen er ikke skalerbart, og i det generelle tilfellet derfor ingen god løsning. Imidlertid bruker jeg en slik løsning for å slippe å sende store mengder med informasjon mellom sidene i applikasjonen, da jeg behøver informasjon om applikasjonsbrukeres venner videre i tjenesten. I en større implementasjon ville det vært ideelt å lagre informasjonen som objekter i en array, hashmap eller annen type datastruktur. Man kan alternativt også opprette en database med passende tabeller som holder på informasjonen for oss.

Informasjon om applikasjonsbrukerens venner fremvises ikke i index-siden, men lagres i en fil for brukeren. Selv om vi «bak kulissene» lagrer denne informasjonen, så bør applikasjonsbrukeren også selv få fremvist en oversikt over vennene sine. I siden *friends.php*, så vil den lagrede FacebookFriendFile-filen åpnes, og innholdet vil bli fremvist for brukeren. Enhver bruker av applikasjonen vil ha en slik fil, da den opprettes første gang i index-siden, og oppdateres ved hver innlogging i applikasjonen. I tillegg til navn og Facebook-ID som finnes i filen, så vises også vennenes bilder, ved å forespørre *https://graph.facebook.com/<Venners FBID>/picture*, med vennenes Facebook-ID. Linken til friends.php-siden finnes øverst på index-siden, under *My Contacts*.

```
echo "<a href='http://sonamr.at.ifi.uio.no/php/Master/friends.php?fbid=$uid&person=$name'>
My Contacts </a>";
```

Fremvisningen av vennene i siden ser slik ut:

Figur 7.8: VoIPBook: Side med informasjon om applikasjonsbrukerens venner.

For å se hvordan informasjon hentes fra FacebookFriendFile-filen og presenteres i siden, se *friends.php* under Vedlegg A.

7.4.3 SIP kontaktinformasjon

Hittil har vi sett på applikasjonsdata som vi kan hente ut fra graf-APIet, som skal gi applikasjonsbrukeren en følelse av at han har en egen profilside, og en side med informasjon om hans venner.

For at applikasjonen skal brukes som et kontaktoppslagsverk for SIP-adresser, så må vi se på hvordan slik informasjon kan oppgis, lagres og gjenopphentes. Det finnes forskjellige måter å utføre dette, da dette er informasjon som ikke kommer fra graf-APIet, men er informasjon som brukere skal oppgi. I applikasjonen har jeg valgt å la brukere oppgi og registrere sine SIP-adresser til applikasjonen, via et PHP skjema. Et slikt skjema er enkelt å konstruere, og samtidig også enkelt for applikasjonsbrukere å benytte. Alternativt kunne man bruket et popup-vindu, en knapp og lignende komponenter. Brukere kan logge inn og benytte applikasjonen selv om de ikke oppgir noen kontaktadresser. Imidlertid er formålet med kontaktoppslagsverket at brukere skal oppgi sin SIP-kontaktinformasjon, og at de skal kunne ringe hverandre på grunnlag av de adressene de oppgir.

For å oppgi en kontaktadresse, vises et tekstområde til brukere på indexsiden, hvor de kan taste inn kontaktinformasjonen og sende det inn til applikasjonen. Når brukeren taster inn en kontaktadresse i tekstområdet, vil det også her opprettes en fil for brukeren, hvor det brukerens oppgitte kontaktadressen lagres. Filen kalles *FacebookAdressFile* og er også konkatenert med brukerens Facebook-ID for å entydig identifisere hvem applikasjonsbrukers kontaktadresser som ligger lagret i filen. Bruker med Facebook-ID «100000852162503» får «FacebookAdressFile-100000852162503.txt» som adressefil. Adressefilen ligger også på samme webområde og i samme mappe som index-filen. Dersom brukeren tidligere har oppgitt en kontaktadresse og dermed har en slik adressefil, så vil nye oppgitte adresser legges til den eksisterende adressefilen. Dersom det er første gang brukeren oppgir adressen, vil en ny adressefil for brukeren opprettes, og adressen vil bli skrevet til filen:

```
$inserted = $_POST['insert'];
```

```
<if (!isset($_POST['submit'])) {

<div id="Insert URI">
<form name="InsertURI" action="<?php echo $PHP_SELF;?>" method="post">
<textarea rows="4" cols="50" name="insert" wrap="physical">
Enter your SIP-URI </textarea>
<br/>
  <input type="submit" value="Submit" name="submit" />

else {

  $myAdrFile = "FacebookAdressFile" ."-" . $me['id'] . ".txt";
  $existingHandle = fopen($myAdrFile, 'a');

  if ($existingHandle != FALSE) {
    fwrite($existingHandle, $inserted . "\n");
    fclose($existingHandle);
  }

  else {
    $newHandle = fopen($myAdrFile, 'w') or die("can't open file");
    fwrite($newHandle, $inserted . "\n");
    fclose($newHandle);
  }
  echo "You have successfully inserted following contact-adress: "
 . $inserted;
}
```

Tekstområdet for brukeren ser slik ut i profil-siden:

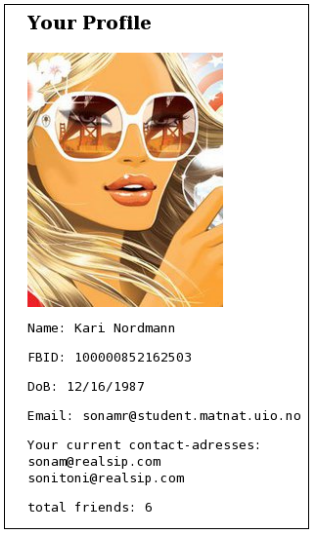
Figur 7.9: VoIPBook: Applikasjonens tekstområde for å oppgi kontaktadresser.

I det brukeren logger inn i applikasjonen, gjør jeg også en sjekk om brukeren ved tidligere innlogging har oppgitt noen kontaktadresser, ved å sjekke om det finnes en FacebookAddressFile med hans Facebook-ID. Dersom det eksisterer en slik fil og brukeren har oppgitt kontaktinformasjon tidligere, skrives adressene også ut på index-siden:

```
$existingFile = "FacebookAdressFile" ."-" . $me['id'] . ".txt";
$existingHandle = fopen($existingFile, 'r');
```

```
if ($existingHandle != FALSE) {
  $contents = fread($existingHandle, filesize($existingFile));
  echo 'Your current contact adresses: ' . "\n";
  echo $contents;
  }
fclose($existingHandle);
```

I testbrukeren min (Kari Nordmann), har jeg oppgitt to SIP-adresser som mine kontaktadresser. Kontaktadressene er tildelt av *Paradial* til testingsformål i oppgaven. Jeg har gitt testbrukeren min disse to adressene som hennes kontaktadresser i applikasjonen:



Figur 7.10: VoIPBook: Brukerens index-side med oppgitte kontaktadresser.

På samme måte som med filen over applikasjonsbrukerens venner, så er det heller ikke skalerbart å ha en fil som holder på hver brukers kontaktadresser. Dersom man i en større implementasjon for eksempel velger å holde på applikasjonsinformasjon i en database, kan også brukeres kontaktadresser legges inn som et felt i forekomstene.

Også i siden som viser informasjon om applikasjonsbrukerens venner, så sjekkes det om applikasjonsbrukeren har oppgitt noen kontaktadresse. Dersom han har oppgitt noen adresser, så blir disse skrevet ut før informasjon om brukerens venner skrives ut. I siden gjøres det imidlertid den viktigste sjekken. Ettersom at vi har Facebook-IDen til alle av brukerens venner, kan vi sjekke om det finnes noen adressefil til vennene av brukeren. På den måten så får applikasjonsbrukeren også innsyn i vennenes kontaktadresser, dersom de har oppgitt noen. Dersom en venn av brukeren ikke har oppgitt noen slik adresse enda, så vil det ikke bli skrevet

ut noen kontaktinformasjon om vedkommende, kun generell informasjon som navn og Facebook-ID.

Sjekken gjøres i en metode, *findAdr()* som tar i mot en Facebook-ID som parameter, og sjekker etter en adressefil som gir treff på ID-en. Dersom det er funnet et treff, skrives adressene i filen til siden. Denne sjekken gjøres for alle av brukerens venner. En slik sjekk gjør at en applikasjonsbruker ikke får tilgang til alle applikasjonsbrukeres eventuelt oppgitte adresser, men kun adresser til sine venner:

```
  ....

function findAdr($fbid) {

  ....

  $frFile = "FacebookAdressFile" ."-" . $fbid . ".txt";
  $frHandle = fopen($frFile, 'r');

  if ($frHandle != FALSE) {
    $contents = fread($frHandle, filesize($frFile));
    echo " contact: ";
    $adrs = explode("\n", $contents);
    foreach ($adrs as $adr) echo $adr . " ";
  }
  fclose($frHandle);
}
```

Siden ser slik ut for Kari Nordmann, der hun har to venner som også er brukere av VoIPBook-applikasjonen, og som har oppgitt sine kontaktadresser:

Figur 7.11: VoIPBook: Side med informasjon om venner og eventuelt deres kontaktadresser.

I VoIPBook-applikasjonen kan brukere nå se generell informasjon om seg selv og sine venner, oppgi sine SIP-adresser til andre venner og se andre venners SIP-adresser, som en brukbar kontaktoppslagstjeneste. For å få flere brukere til å benytte denne tjenesten, tillater jeg applikasjonsbrukere

å oppfordre sine venner til å bruke tjenesten. Applikasjonen kan publisere en tekst til applikasjonsbrukerens vegg, som ber venner av brukeren om å også ringe med VoIP via VoIPBook.

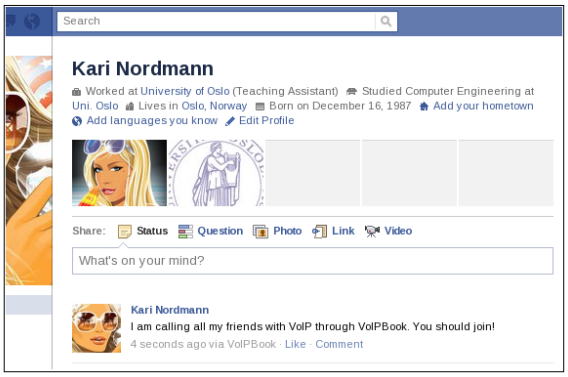
Publiseringen til applikasjonsbrukerens vegg gjøres med en HTTP POST forespørsel, «https://api.facebook.com/method/stream.publish?». Forespørselen tar beskjeden som skal publiseres på applikasjonsbrukerens vegg i parameteren, *message*, og Facebook-ID'en til brukeren i parameteren, *uid*. Aksessloddet må også gis med i forespørselen:

```
$message = "I am calling all my friends with VoIP through VoIPBook. You should join!";
$invite = "https://api.facebook.com/method/stream.publish?
message=" . $message . "&uid=" . $uid . "&access_token=" . $token;
```

I index-siden finnes det en link, «Invite your Friends», som utfører publiseringen ovenfor:

```
echo '<a href="' . $invite . '"> Invite your Friends  </a>';
```

Publiseringen vil se slik ut på applikasjonsbrukerens side:



Figur 7.12: VoIPBook: Publisering til applikasjonsbrukeres vegger for å oppfordre flere brukere å benytte applikasjonen.

Nå som kontaktoppslagstjenesten er laget, så er det videre på tide å se på hvordan brukerne kan ringe med informasjonen de har fått fra VoIPBook. I oppgaven har jeg valgt å bruke en SIP-klient som skal ta seg av selve signaleringen og mediaoverføringen i VoIP-sesjonen. Det vi må gjøre er å se på hvilke muligheter vi har til å initiere en VoIP-samtale ved å starte SIP-klienten og opprette en sesjon med ønsket bruker, fra VoIPBook. I oppgaven bruker jeg SIP-klienten, *SJPhone*, og i neste kapittel skal jeg kort introdusere denne klienten.

For fullstendig kode av applikasjonens oppslagstjeneste, se *index.php* og *friends.php* under Vedlegg A.

Kapittel 8

SIP-klienten: SJPhone

8.1 Om SJPhone

SJPhone er en softtelefon fra softtelefon-produsenten, *SJ Labs*. SJ Labs produkter har vært anvendt i over 5 år, og softtelefonene har blitt nedlastet og brukt av over 2 millioner brukere [25].

SJPhone tillater brukere å kommunisere over Internett eller er annet IP nettverk med desktop PCer, PDAer, frittstående IP-telefoner og også tradisjonelle fasttelefoner og mobiltelefoner [26]. SJPhone støtter både SIP og H.323 standardene, mediaoverøfringsprotokollene RTP og RTCP, og NAT-traverserings protokollen for multimedia, STUN. SJphone er fullt kompatibel med de største internasjonale Internett-telefoni tjenesteleverandørene *(Internet Telephony Service Providers (ITSP))*[1] og VoIP software og hardware produsenter, som Cisco. I tillegg til å være en klient som tilbyr brukere å ringe VoIP, er SJphone er også en Chat-klient som tilbyr sanntids-direktemeldinger og tilstedeværelse[26]. SJPhone er en åpen og kostnadsfri programvare, og er tilgjengelig for følgende plattformer:

Windows 2000

Windows XP

Windows Vista

Mac OS X

Linux

Windows Mobile 5

For å gi brukere god kvalitet på VoIP-sesjoner, tilbyr SJPhone noen tiltak for å håndtere pakketap, jitter og forsinkelser. SJphone er som standard konfigurert til å ha en pakketapsrate på under 10-15 prosent. Brukere har muligheten til å bli indikert om hvor mange pakker som går tapt, under en

[1] En ITSP tilbyr digitale telekommunikasjonstjenester som er basert på VoIP, og som tilbys via Internett.

sesjon. SJPhone har også et buffer for å håndtere forsinkelser og jitter. Jitterbufferet er initielt satt til å holde på seks pakker og forsinkelsesintervallet er på 20ms, men begge kan justeres manuelt av brukeren. Brukeren kan i tillegg justere hvor mange pakker I/O-køene skal holde. SJphone tilbyr brukere også muligheten til å se antall pakker som kommer i gal rekkefølge under en sesjon [26].

SJPhone støtter kodekene [2], *G711 A -Law*, *G711 u -Law*, *GSM*, *G.729* og *Internet Low Bit Rate Codec (iLBC)*. Som med SJPhones interne buffere og køer, står brukeren fritt til å velge hvilken kodek han ønsker å bruke til koding/dekoding og kompresjon av audio i sesjonen. Valg av kodek gjøres ofte i henhold til hvordan nettverk brukeren befinner seg på, og lokasjonen til samtalepartneren. G711 tilbyr god lydkvalitet men liten lydkompresjon, og kan derfor anbefales for nettverk med vid båndbredde, som lokale datanett (Local Area Networks (LANs)). GSM og G.729 kodekene kan anbefales dersom båndbredden i nettverket er begrenset og må bevares, slik som ved langdistanse samtaler eller *dial up*-forbindelser, som i vidstrakte nett (Wide Area Networks (WANs)). G.729 tilbyr god lydkompresjon, GSM tilbyr gjennomsnittlig lydkvalitet og bra lydkompresjon. iLBC tilbyr også god lydkvalitet, og kan anbefales i nettverk med stort pakketap [26].

8.2 Hvorfor SJPhone?

I en tabell tatt fra *International Telecommunication Union*, anbefalning G.114, vises det en oversikt over VoIP-forsinkelser som regnes som akseptable:

- Loss should be no more than 1 percent.
- One-way latency (mouth to ear) should be no more than 150 ms.
- Average one-way jitter should be targeted at less than 30 ms.

Figur 8.1: Tabell av ITU over akseptable VoIP-forsinkelser.

Som det går frem, så bør pakketap være mindre enn 1 prosent av totalt antall pakker som mottas. En akseptabel ènveis forsinkelse er satt til å være maksimalt 150 millisekunder, altså fra da oppringeren snakker, til mottakeren hører. Akseptabel ènveis jitter er satt til mindre enn 30 millisekunder i gjennomsnitt av samtalen.

I sammenheng med et prosjekt om VoIP-ytelse, har jeg tidligere gjort en sammenligning mellom SJPhone og *Skype*[3], med hensyn på pakketap, jitter og forsinkelser. Forsøket gikk ut på å overføre en lydfil fra en bruker, A,

[2] En kodek er et program eller enhet som koder og/eller dekoder digitale datastrømmer eller signaler. En kodek koder for transmisjon, lagring eller kryptering, og dekoder for avspilling eller redigering.

[3] Skype er en programvare som tillater brukere å ringe med VoIP og chatte over Internett.

til en annen bruker, B, som befant seg på samme lokale datanett. Overføringen ble utført i to forsøk, en gang med SJphone og en gang med Skype. Under overføringen brukte jeg protokollanalyseringsverktøyet, *Wireshark*, for å si noe om klientenes pakketap, forsinkelse og jitter. Etter å ha analysert dataene fra Wireshark, kom jeg frem til følgende resultater:

	Akseptabel	SJPhone (A til B)	SJPhone (B til A)	Skype (A til B)	Skype (B til A)
Antall pakker mottatt:		4016	4026	1680	4637
Pakketap:	1%	0%	1 pakke(0%)	0%	4 pakker(0.001%)
Forsinkelse (èn vei):	<150ms	82.41ms	207.68ms	2ms	64,5ms
Jitter:	<30ms	3.95ms	32.29ms	44ms	57ms

Tabell 8.1: Ytelsessammenligning: SJphone og Skype.

Som det går frem av tabellen, så er jitter-verdien ganske høy for SJPhone, og også forsinkelsen i den ene retningen, er utenfor det som er anbefalt for VoIP.

SJphone-versjonen jeg har benyttet er en kostnadsfri versjon, og til å være en kostnadsfri programvare opplevdes kvaliteten som forholdsvis bra i gjennomsnitt under samtalen. SJphone kommer også i en premiumversjon som ikke er kostnadsfri, hvor jeg antar at kvaliteten er bedre. Imidlertid er resultatene ovenfor gjort uten å ha utført noen endringer i SJPhones audio-innstillinger for optimalisering. SJphone er i tillegg en kryssplattformkompatibel programvare som er åpen og er testet til å fungere på Windows, Mac og Linux. På grunnlag av dette velger jeg å bruke SJPhone i denne oppgaven.

Det finnes en rekke andre åpne og kryssplattformkompatible SIP-klienter som er godt kvalifiserte, men ettersom klienten kun trengs til demonstrasjonsformål i oppgaven, så har jeg valgt en klient som jeg har benyttet tidligere, og som jeg vet fungerer. Vi skal senere se at hvilken SIP-klient som brukes ikke har så veldig stor betydning i denne applikasjonen.

Kapittel 9

Tilnærminger for å starte SJPhone fra VoIPBook

Til nå har vi fått opp en Facebook-applikasjon som fungerer som en kontaktoppslagstjeneste for SIP-adresser. Vi har også funnet en SIP-klient som er gratis og kryssplattformkompatibel, og kan utføre anropsprosidyren. En forutsetning for at brukeren skal kunne ringe med VoIP via VoIPBook og med SJPhone, er at brukeren har installert klienten på maskinen sin. Vår oppgave er så å starte denne SIP-klienten fra VoIPBook-applikasjonen vår med input fra applikasjonen, som sier hvem SIP-adresse vi ønsker å starte SIP-klienten med.

Selv om innholdet i Facebook-applikasjoner lastes inn for applikasjonsbrukeren via Facebook-plattformen, så vil det være et klient/server-forhold mellom brukeren og applikasjonen. På grunn av sikkerhet og for å verne brukere for skadelig kode, så er det ikke rett frem å utføre operasjoner på klientsiden fra serversiden. Som regel behøves autorisasjon av brukere for å få tilgang til brukerens maskin. Med utgangspunkt i dette, så skal vi videre se på hvilke muligheter som finnes for å overføre innhold og å utføre operasjoner på en klientside, fra en serverside. Vi ønsker i tillegg å finne en løsning som tillater flest mulige brukere å benytte applikasjonen, for å best mulig utnytte nettverkseffekten som eksisterer i Facebook.

9.1 Nettleser plugins

VoIPBook-applikasjonen er testet og fungerer med nettlesere som Internet Explorer, Firefox, Opera, Safari og Google Chrome. For å finne en måte å overføre informasjon fra et nettsted til en klientmaskin, fikk jeg et tips om å se på å se på hvilke *browser-plugins* som finnes i nettlesere. En nettleserplugin brukes synonymt med *add-on* og *extension*, og er et sett med programvarekomponenter som legger til spesifikke funksjoner og kapasiteter til en større programvare applikasjon. Slike plugins er vanlig å bruke i nett-

lesere for direkteavspilling av filmer, fremvisning av bilder, animasjoner, filer og lignende. Noen plugins er som en standard lagt til i nettleseren, og for de fleste nettlesere er dette plugins som:

Adobe Reader: For å fremvise og skrive ut Adobe PDF

Adobe Flash Player: For levering av Adobe Flash opplevelser

Java Platform SE 6: For å tillate maskinen å kjøre Java applikasjoner og appleter

QuickTime: For å spille/se/interagere med video/audio/VR/grafikk filer

RealPlayer: For å spille streaming innhold som sanntids video/audio

Shockwave: For å fremvise webinnhold som har blitt generert med Adobe Director

Windows Media Player: For å spille innebygd audio/video fra webside gjennom plugin.

Utover de innebygde plugins, så tilbyr de fleste nettlesere et hav av ytterligere plugins sortert etter kategori, som brukeren kan legge til i nettleseren sin etter behov. De fleste nettlesere tilbyr også tredjepartsutviklere muligheten til å utvikle egendefinerte plugins til nettleseren.

Å lage en egendefinert plugin kan være et alternativ for å gi input til SJPhone fra VoIPBook. Plugin'en kunne eksempelvis lese kontaktinformasjon fra siden til VoIPBook-applikasjonen, lagret dataene i et akseptabelt filformat og la brukeren laste det ned på sin lokale maskin for fremvisning. En nettleser-plugin kjører typisk i samme prosess som nettleseren, og har dermed som regel samme rettigheter som nettleseren. Vi ville dermed kunne startet en annen applikasjon som SJPhone, fra plugin'en.

En ulempe med denne løsningen er at en plugin er nettleseravhengig. Hvis vi skal lage en løsning med en nettleser-plugin, så er begrenser vi brukere til å benytte VoIPBook via den bestemte nettleseren vi har laget plugin for. Alternativt kunne man laget en slik plugin til flere nettlesere, som er en tungvinn løsning da utvikleren må studere alle nettleseres APIer. Det er også en kontinuerlig jobb ettersom nettlesere stadig oppgraderes og kan endre APIene, og løsningen må gjøres for alle nye nettlesere som utvikles. I oppgaven ønsker vi å ha en løsning som ikke begrenser brukere for mye og tillater størst mulig brukermasse å benytte applikasjonen, for å få størst mulig økning i nettverkseffekt for VoIP-tjenestene vi implementerer. En plugin-løsning er derfor ikke ideell.

9.2 Cookies

Et annet alternativ er å se på nettleseres *Cookies*. En cookie er en variabel som lagres på brukerens lokale maskin, av en nettside. Hver gang maskinen forespør samme nettside i en nettleser, så vil nettleseren sende med den lagrede cookie-variabelen også. Cookie-variabler er brukt for å huske informasjon om brukeren som kan være interessant for en nettside. Eksempler på slike cookie-variabler, kan være:

Navne cookie: Første gangen brukeren besøker nettsiden vil han bli bedt om å oppgi navnet sitt. Navnet lagres i en cookie. Neste gang brukeren besøker siden kan man få en personlig velkomst beskjed, etter at navnet er gjenopphentet fra den lagrede cookie-variabelen.

Passord cookie: På samme måte så vil brukeren bli bedt om å oppgi et passord til nettsiden første gang han besøker den. Dette passordet kan bli husket og lagret i en cookie-variabel, slik at brukeren ikke trenger å oppgi det neste gang han besøker siden.

De fleste nettlesere tilbyr også tredjepartsutviklere muligheten til å utvikle og sette egendefinerte cookies. Et alternativ kunne derfor vært å ha en cookie-variabel en brukermaskin som lagrer kontaktadresser til venner av brukeren, hentet fra VoIPBook, eksempelvis i et string-array. For hver gang det blir lagt til eller fjernet en kontaktadresse i Facebook-applikasjonen, så kan cookie-variabelen oppdateres deretter. Cookies lagres for hver bruker på en maskin, så dersom en annen bruker logger på samme maskinen rett etterpå og går inn på samme webside, så vil nye cookies opprettes og lagres for denne brukeren. En slik variabel kan enkelt opprettes i menyen til nettlesere, og er mindre tungvint å implementere enn nettleser-plugins.

Ulempen med denne løsningen i likhet med plugin-løsningen, er at cookies som opprettes i nettleserens meny også er nettleseravhengig. For at applikasjonsdataene skal overføres til klientens maskin, så må utvikleren i såfall opprette en slik cookie-variabel i menyen til hver nettleser. Alternativt kan man ganske enkelt opprette, lese og slette cookie-variabler i et dokument med JavaScript. På den måten så trenger ikke utvikleren eksplisitt å lage en cookie for hver nettleser, JavaScript-koden vil gjøre det i hver nettleser som åpner dokumentet. Imidlertid så er en cookie kun en variabel som holder på en bestemt type verdi, og kan heller ikke på noen måte starte opp en applikasjon på klientsiden. Formålet med cookies er imidlertid å gi hukommelse til webservere og nettlesere om brukeren selv. Å utnytte en slik variabel til å holde på Facebook applikasjonsdata, ville vært en mindre elegant løsning.

9.3 Client-side-skripter

Client-side-skripter er skripter som enten er innebygd i et HTML-dokument *(embedded script)* eller som ligger i en separat fil, men refereres til av HTML-dokumentet *(external script)*. Når en handling aktiverer skriptet, så vil filer nødvendig for å utføre skriptet bli sendt over til brukerens maskin, av serveren filene ligger på. Brukerens nettleser vil så eksekvere skriptet på klientsiden, og fremvise HTML-dokumentet med eventuell output fra skriptet. Eksempler på lokalt eksekverbare skript som dette, kan være *Client-side JavaScript* og *Client-side Visual Basic Script (VBScript)* [7].

Slike client-side-skripter åpner muligheten for å definere et sett med operasjoner i et egendefinerte skript, som skal utføres lokalt på brukerens maskin. Med et slikt skript kunne vi skrevet kommandoer for å lansere

en brukerapplikasjon, som SJPhone, og til og med gitt den input. Ulempen med slike skripter er at utvikleren som regel ikke får tilgang til brukerens maskin, utover til nettleserapplikasjonens funksjoner på maskinen. På grunn av sikkerhetsbegrensninger så vil utvikleren av skriptet ha veldig begrenset tilgang, slik at selv om vi kan skrive kommandoer som gjør det vi ønsker, så vil de ikke utføres på grunn av mangel på aksess. Det kreves også at nettlesere støtter skriptspråket skriptet er skrevet i, for at det skal kunne kjøres.

9.4 ActiveX kontroller

ActiveX Controls er en del av *Microsoft ActiveX*-teknologien, som er et rammeverk for å definere gjenbrukbare programvarekomponenter uavhengig av programmeringsspråks[68]. ActiveX-kontroller er små programmer eller byggeblokker som er til for å lage distribuerte applikasjoner over Internett, via nettlesere. Et eksempel på ActiveX kontroller er egendefinerte applikasjoner for innsamling av data, fremvisning av spesielle type filer og animasjon, og å utføre operasjoner. Kontrollene kan skrives i programmeringsspråk og miljøer som C++, Delphi, Visual Basics og .NET rammeverket [35].

ActiveX-kontroller ble utviklet for å tillate nettlesere å laste kontrollene ned, og eksekvere dem. Imidlertid kan ActiveX-kontroller kun operere på Windows-plattformen, med en Internet Explorer-nettleser. Selvom ActiveX-kontroller er operativsystem- og nettleserbegrenset så kan de, i motsetning til alternativene vi har studert hittil, oppnå full aksess til operativsystemet. Dette betyr at når kontrollen først er lastet ned på brukerens maskin, kan den utføre hvilken som helst funksjon som utvikleren måtte ønske.

En slik kontroll gir mulighet for å skrive en programblokk som tar input fra VoIPBook, og lanserer SJPhone med inputen. Nettleseren ville vært i stand til å nedlaste kontrollen, og eksekvere programkoden. Dette er en løsning som kan fungere i implementasjonen vår, men ettersom at ActiveX-kontroller begrenses kun til Windows-plattformen og Internet Explorer, og brukermassen dermed begrenses til Windows-brukere, så har jeg valgt å ikke bruke denne løsningen i oppgaven min.

9.5 Applets

En *Applet* er en liten applikasjon som utfører en spesifikk oppgave eller en eller flere enkle funksjoner, noen ganger i kontekst av et større program, og muligens som en plugin [78]. Noen applets kan fungere som frittstående applikasjoner opp av et operativsystem. Slike applets er imidlertid små i størrelsen og utfører kun et begrenset sett operasjoner, og er ikke et

fulltfunksjonelt applikasjonsprogram. I andre tilfeller så kan applets ikke kjøres på en uavhengig måte, men fungerer som en utvidelse av annen programvare. Slike applets må kjøre i en *Container* [1], som opprettholdes av et vertsprogram eller andre type applikasjoner, gjennom en plugin [78]. Imidlertid så refererer begrepet ofte til *Java Applets*, som er småprogrammer skrevet i Java-programmeringsspråket og som kan inkluderes i websider [78].

9.5.1 Java applets

En Java-applet er et program som er skrevet i Java-programmeringsspråket, og som kan inkluderes i en nettside. Slike applets kan tilby interaktiv funksjonalitet til web-applikasjoner som ikke kan tilbys av HTML alene. I motsetning til *Servlets*, så eksekverer applets kun i klientens miljø av et system, da servlets eksekvere på serversiden. Java-applets skrives i *Java bytecode* og ettersom Javas bytecode er plattformuavhengig, kan koden eksekveres av nettlesere under forskjellige plattformer, som Windows, Unix, Mac OS og Linux. Når en Java-aktivert nettleser prosesserer en side som inneholder en applet, så vil applet-koden overføres fra serveren den ligger på til klientens system, og bli eksekvert av nettleserens *Java Virtual Machine (JVM)*. Java applets er som regel skrevet i Java programmeringsspråket, men kan også skrives i andre programmeringsspråk som kompilerer Java bytecode, slik som *Jython*, *JRuby* eller *Eiffel* [73, 45]. I Java Platform SE 6, så finnes blant annet pakken *Java.applet* som brukes til å bygge slike Java applets [36].

Etter at applet-koden er lastet ned fra en webserver som holder på koden, kan nettleseren fremvise innholdet i nettsiden, eller åpne et nytt vindu som fremviser applet'ens brukergrensesnitt. Applet'en kan fremvises som en innebygd del av nettsiden ved å benytte HTML-elementet, *applet*, eller *object*. *Sun* anbefaler bruk av applet-elementet for anvendelse i flere nettleser miljøer [73]. Nettleseren må være en Java-aktivert nettleser for at applet'en skal kunne lastes inn, eksekveres og fremvises [39].

9.5.2 Java-applet sikkerhetsmodeller

Java-applets forekommer i en av tre typer sikkerhetsvarianter, med veldig forskjellige sikkerhetsmodeller: *unsigned applets*, *signed applets* og *self-signed applets*.

En usignert applet, også omtalt som upålitelig applet, har en rekke begrensninger og restriksjoner grunnet dens upålitelighet. Usignerte applets har ingen aksess til det lokale filsystemet på klientsiden, og har webaksess begrenset til applet'ens nedlastningsside. Upålitelige applets kan ikke

[1] En container er en klasse, datastruktur eller abstrakt datatype hvis instanser er samlinger av andre objekter. Containere brukes til å lagre objekter på en organisert måte, etter spesifikke regler

aksessere systemegenskaper, kalle på intern kode, eller eksekvere eksterne kommandoer på det lokale systemet. Dersom usignerte applets kjøres i en egen ramme, så vil rammen ha en header som indikerer at applet'en er upålitelig.

En signert applet, også referert til som en pålitelig applet, kan verifisere gjennom en uavhengig sertifisieringsautoritær server *(certificate authority server)*, og kan bevise verifiseringen ved å få en *signatur*. For å produsere en slik signatur trengs spesialisert verktøy og interaksjon med autoriseringsserverens opprettholdere. Når signaturen er verifisert og brukeren av maskinen også innvilger, så kan en signert applet få ytterligere rettigheter, ekvivalent med et normalt frittstående program på brukerens maskin. Rasjonaliseringen er at forfatteren av appleten er kjent, og vil holdes ansvarlig for bevisst skade. Denne tilnærmingen tillater applets å bli brukt til mange oppgaver som vanligvis ikke ville vært mulig med client-side skripting. Derimot krever denne tilnærming også mer ansvar fra brukeren, da han selv må bestemme hvem som er pålitelige og eller ikke ettersom aksessen er avhengig av brukerens innvilgelse.

Selvsignerte-applets er applets som er signert av applet-utvikleren selv, og ikke av en uavhengig tredjepart som autoriseringsserveren i signerte-applets. Java-plugin'en i nettleseren gir en advarsel til brukeren når en selvsignert applet skal autoriseres, ettersom at applet'ens funksjon og sikkerhet er garantert av utvikleren alene, som en følge av at han ikke blitt verifisert av en sertifiseringsautoritet. Dette gjør at brukeren må være enda mer ansvarlig og oppmerksom på hvilke applets han avgjør som pålitelige.

9.5.3 Fordeler og ulemper med Java-applets

Fordelen med å bruke Java-applets er at de enkelt kan gjøres kryssplattformkompatible, og at de støttes av de vanligste nettlesere. En og samme applet kan fungere på alle installerte versjoner av Java samtidig, i stedet for kun den siste Java plugin-versjonen. De fleste nettlesere *cacher* applets, slik at den rask vil lastes inn når man returnerer til nettsiden. Java-applets kan også flytte arbeid fra serversiden til klientsiden, slik at web-løsningen er mer skalerbar med et høyt antall brukere. Slike applets kan på en naturlig måte støtte endringer i brukertilstand, og utviklere kan utvikle og debugge appleten ved å bruke *main()*-metoden. Sun tilbyr også *AppletViewer* som er et frittstående kommandolinjeprogram og verktøy som kan brukes til å testkjøre Java-applets. En upålitelig applet har ikke noen aksess til den lokale maskinen, som gjør en slik applet mindre truende for brukeren. På den annen side så kan en pålitelig applet ha full aksess til maskinen den kjører på, dersom brukeren innvilger det. Java-applets er raske og kan ha lignende ytelse som programvare allerede installert på maskinen [73].

Ulempen med Java-applets er at de krever at nettleseren har en Java-plugin. Noen organisasjoner tillater kun programvare som er installert av

en administrator, og det kan derfor være en utfordring å rettferdiggjøre installasjon av Java-plugin. På grunn av kan sikkerthetsrestriksjoner kan det være vanskelig og muligens umulig for en upålitelig applet til å oppnå de ønskede målene. Noen applets krever et spesifikk *Java Runtime Environment (JRE)*, og dersom en applet krever et slikt JRE eller et nyere JRE enn det som er tilgjengelig i nåværende Java-versjon i systemet, så må brukeren vente inntil det JRE er ferdig nedlastet, som kan ta tid.

Etter å ha sett på noen av tilnærmingene som finnes for å overføre innhold og utføre handlinger fra en serverside til en klientside, så har jeg i oppgaven valgt å benytte en Java-applet. Ettersom at Java-applets kan bygges inn i nettsider, så kan vi enkelt lage en side i VoIPBook-applikasjonen som laster inn og fremviser applet-koden. Java-applets er kryssplattformkompatible og er en løsning som fungerer på flere plattformer og nettlesere, og de nettleserne jeg har testet VoIPBook med, har en plugin for Java. I VoIPBook-applikasjonen kan vi overføre data fra andre sider til appletsiden, og vi har potensial for å oppnå aksess som tillater å starte SJPhone-klienten. Ettersom Java-applets er en løsning som kan utføre den jobben vi ønsker, og samtidig tillatter brukere med forskjellige plattformer og nettlesere, så vil jeg si at slike applets er en løsning som i denne implementasjonen dekker størst brukermasse, og kan gi størst nettverkseffekt. I neste eksempel skal vi se på appleten, *AppLauncher*, som lar VoIPBook starte SJPhone, og tillater applikasjonsbrukerne å ringe med VoIP.

Kapittel 10

Java-applet'en: AppLauncher

10.1 Oppsett av Applet'en

For å åpne SJPhone på brukerens maskin via VoIPBook, har vi bestemt oss for å bruke en Java-applet. Appleten skal vises i en egen side i VoIPBook-applikasjonen. Vi ønsker at appleten skal presentere applikasjonsbrukeren med navn og SIP-adresser til venner han kan ringe til. Brukeren skal så kunne velge hvem venn han ønsker å ringe med SIP-klienten, og den valgte vennens SIP-adresse skal være input til SIP-klienten. Appleten skal videre starte SIP-klienten og gi klienten inputen brukeren har valgt.

Linken til applet-siden er øverst på index-siden til applikasjonen, med teksten «Place a Call». Når en bruker trykker på linken, vil han omdirigeres til filen *applet.php* som inneholder applet-koden. Denne filen ligger på samme filområde som index-filen til applikasjonen. Applet.php mottar Facebook-ID og navn til applikasjonsbrukeren som URL-parametere:

```
echo "
<a href='http://sonamr.at.ifi.uio.no/php/Master/applet.php?
fbid=$uid&person=$name'>
Place a Call </a>";
```

I applet-siden så skrives navnet til applikasjonsbrukeren ut, samt brukerens profilbilde. Videre så leses FacebookFriend-filen konkatenert med applikasjonbrukerens Facebook-ID, for å hente ut alle vennene til brukeren. For hver venn av brukeren som finnes i filen, så sendes navnet og ID'en til en metode *findAdr($fbid, $name)*. Denne metoden skal sjekke for alle venner av brukeren om de har oppgitt noen kontaktadresser. Dette gjøres ved å sjekke om det finnes en FacebookAdress-fil til Facebook-ID'en som metoden mottar som parameter. Metoden kalles altså med alle av applikasjonsbrukerens venners Facebook-ID'er:

```
$existingFile =  "FacebookFriendFile" ."-" . $id . ".txt";
$existingHandle = fopen($existingFile, 'r');

if ($existingHandle != FALSE) {

  $contents = fread($existingHandle, filesize($existingFile));
  $words = explode(";", $contents);
  foreach ($words as $word) {
    $tmp = explode(":", $word);
    $fbid = trim($tmp[0]);
    $name = tmp[1];
```

```
    findAdr($fbid, $name);

  }

  fclose($existingHandle);
 }

else echo "Cannot open file.";
```

Dersom det finnes en slik FacebookAdressFile-fil for en venn av brukeren, så skal SIP-adressen(e) og navnet til vennen tas vare på, og presenteres for brukeren som en mulighet til å starte en VoIP-samtale mot. Jeg har opprettet to globale arrayer i applet.php som skal holde på denne informasjonen: *friends* holder på navnet til vennene, og *friends_adrs* holder på adressene til vennene. I applet'en er vi kun interessert i å presentere venner som har oppgitt kontaktinformasjon. En venn kan ha oppgitt flere SIP-adresser, og applikasjonsbrukeren må bli presentert med muligheten til å starte en VoIP-samtale med enhver av disse adressene. findAdr-metoden ser slik ut:

```
function findAdr($fbid, $name) {

  ...

  $frFile = "FacebookAdressFile" ."-" . $fbid . ".txt";
  $frHandle = fopen($frFile, 'r');

  if ($frHandle != FALSE) {
    $contents = fread($frHandle, filesize($frFile));
    $adrs = explode("\n", $contents);
    foreach ($adrs as $adr)  {

        ...

$friends[$num_fr++] = $name . " ";
$friends_adrs[$counter++] = $adr;

    }
  }
   fclose($frHandle);
}
```

Etter å ha samlet informasjonen vi ønsker å gi til appleten, må vi videre skal vi se på hvordan appleten skal settes opp og fremvises.

Som nevnt tidligere så kan applets innebygges i et HTML-dokument med et applet-element eller object-element. Jeg har valgt å bruke applet-elementet på grunnlag av anbefalning fra Sun, ettersom at appleten skal brukes i flere nettlesermiljøer. Applet-elementet har en rekke attributter som blant annet *code*, *archive*, *width* og *height*. Code-attributtet angir filen som inneholder appletens kompilerte subklasse av klassen *Applet*, og er et påkrevd attributt. Filen angis relativt til stien (path) til appleten. Klassefilen som inneholder applet-subklassen kan være på formen *klassenavn.class*, eller *pakkenavn.klassenavn.class*. I mitt tilfelle så er det klassen *AppLauncher* som utvider Applet-klassen, så code-attributtet vil få verdien *AppLauncher .class*.

Archive-attributtet er et opsjonelt attributt, og beskriver en eller flere arkiver som inneholder klasser og andre ressurser som appleten benytter, og som vil ble lastet inn på forhånd. Java-arkivene i arkivlisten er kommaseparert, og arkivene skal være på formen *arkivnavn.jar*. Bredde- og høyde-attributtene angir henholdsvis initiell bredde og høyde i piksler av applet'ens fremvisningsområde, og er påkrevde attributter. Disse attributtene inkluderer ikke vinduer eller dialoger som applet'en eventuelt bringer opp:

```
<P><APPLET code="AppLauncher.class" width="600" height="400">
...
</APPLET></P>
```

Applet-elementet har i tillegg andre opsjonelle attributter som *alt* og *name*. Alt-attributtet angir alternativ tekst som skal fremvises hvis nettleseren forstår applet-elementet, men ikke kan kjøre Java-applets. Name-attributtet spesifiserer navnet til denne applet-instansen, som gjør det mulig for forskjellige applets på samme side å identifisere hverandre, og å kommunisere med hverandre. For en fullstendig liste over applet-elementets attributter, se [67].

Applet-elementet har også et annet viktig attributt, *param*. Med Param-attributtet har man mulighet til å spesifisere en parameter i applet-elementet, som kan aksesseres og hentes ut av applet-subklassen. For eksempel kan param-attributter ha verdier som brukeren oppgir og som sendes til applet-subklassen. Param-attributtet angis på følgende form:

```
<PARAM NAME = appletAttributt1 VALUE = verdi>
<PARAM NAME = appletAttributt2 VALUE = verdi>

...
```

I applet-siden i VoIPBook-applikasjonen har vi funnet navn og SIP-adresser til venner av brukeren som har oppgitt kontaktinformasjon. Denne kontaktinformasjonen ønsker vi å gi med til applet'en, slik at applet'en blant annet kan presentere brukeren med informasjonen på en pen måte. I appleten sender vi denne informasjonen til applet-subklassen ved å bruke param-attributtet.

Venners navn og adresser ligger henholdsvis i friends og friends_adrs arrayene. Jeg går derfor over arrayene i en for-løkke, og lager et param-attributt for hver kontaktadresse som finnes i adresse-arrayet. Parameternavnet lar jeg være telleren «i», i for-løkka som går fra 0 til antall kontaktadresser som er registrert. Parameternavnet legger jeg i variabelen *name* i param-attributtet. Verdien skal være stringen *Place a call to <navn> (<kontaktadresse>)*. Både navnet og adressen er verdiene på i-te plass i arrayene. Verdien legges inn i den tilhørende verdi-variabelen, *value*, i param-attributtet. Siden det er mye tekst i verdifeltet så trenger vi å parsere stringen i applet-subklassen for å hente ut navnet og kontaktadressen, men en slik string gir en fin fremvisning til applikasjonsbrukeren. Antall parameter-attributter som finnes angis i parameteren *counter*. Counter-parameteren sendes til applet-subklassen, slik at den vet hvor mange parametere som skal hentes ut:

```
<P><APPLET code="AppLauncher.class" width="600" height="400">
  <param name="numberOfFriends" value="<?php echo $counter ?>"/>
  <?php for ($i=0; $i<$counter; $i++) {?>
  <param name="display<?php echo $i; ?>"
  value="Place a call to
  <?php echo $friends[$i]; echo '( ' . $friends_adrs[$i] . ' )'; ?>"/>
  <?php } ?>
</APPLET></P>
```

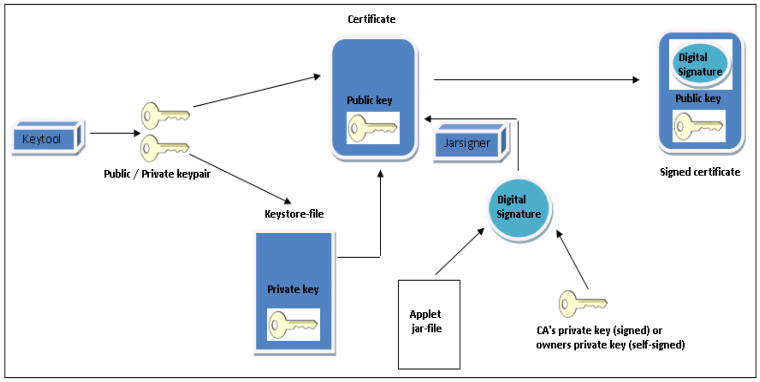
For fullstendig oversikt over oppsett av applet-siden og applet-elementet, se *applet.php* under Vedlegg A.

10.2 Signering av Applet'en

Som vi har sett på tidligere, så finnes det tre forskjellige sikkerthetsvarianter av Java-applets, nemlig signerte, usignerte og selvsignerte applets. Selv om applets eksekveres i iframes, så kan applet'en få tilgang utenfor sine rammer hvis brukeren tillater det. De fleste selskaper bruker signerte applets som er verifisert av en uavhengig tredjepart. Ettersom at applet'en min er til demonstrasjonsformål, har jeg valgt å benytte den enkleste løsningen, som er å lage en selvsignert applet. I selvsignerte applets så vil brukeren få fremvist en advarsel som gjør brukeren oppmerksom på at applet'en er selvsignert, og at utvikleren ikke er autentisert av en tredjepart, men at utvikleren heller er *selvautentisert*. Autentisering innebærer at person eller enitet[1] presenterer en identitet, og et identitetsbevis, som verifiserer at identiteten faktisk tilhører den personen eller entiteten som den påstår å ha. I appleten må jeg på tilsvarende måte autentisere meg for andre entiteter, slik at de får verifisert at det faktisk er jeg som står bak appleten. Dersom brukeren tillater appleten å eksekvere, så inneholder applet'en kode som starter SJPhone med brukervalgt input på hans lokale maskin.

For å utføre en slik selvautentisering, må jeg oppgi en identitet og et bevis på den identiteten. Dette gjøres ved å generere et *kryptografisk offentlig/privat*-nøkkelpar med et tilhørende *sertifikat*. Et slikt nøkkelpar består av en *privat nøkkel* som skal hemmeligholdes, og en *offentlig nøkkel* som kan offentliggjøres. Den offentlige nøkkelen virker som en identitet for en entitet, og kan distribueres. For å lage et identitetsbevis, så brukes den private nøkkelen sammen med en kryptografisk algoritme til å produsere en *digital signatur*. En digital signatur er en string av bits som er en funksjon av dataene som skal signeres, og den private nøkkelen til en entitet. En slik signatur, kan som en håndskrevet signatur, verifisere autentisitet, kan ikke forfalskes hvis den private nøkkelen hemmeligholdes, og kan ikke påstås å være signaturen for annen data, ettersom de signerte dataene determinerer den digitale signaturen. En slik signatur kan heller ikke endres, så dersom den er blitt modifisert eller manipulert, så vil den ikke lenger verifiseres som autentisk. En slik digital signatur bevarer også dataenes *integritet*, i tillegg til entitetens autentisitet, da manipulert data ikke verifiseres og verifisert data ikke kan endres. Når denne digitale signaturen signerer sertifikatet assosiert med det genererte nøkkelparet, får vi et *digitalt signert sertifikat*. Det er dette signerte sertifikatet som brukes som identitetsbevis til entiteten. For mer om offentlig nøkkelpar kryptografi og nøkkelhåndtering, se [28, 29].

[1] En entitet refererer til en person, firma, organisasjon eller lignende, i denne sammenhengen.



Figur 10.1: Offentlig/privat nøkkelgenerering og signering av sertifikat.

I oppgaven bruker jeg nøkkel og sertifikathåndteringsverktøyet, *keytool*. Keytool tillater brukere å administrere deres offentlig/privat nøkkelpar og assosierte sertifikater til bruk for selvautentisering. Keytool genererer et offentlig/privat nøkkelpar, og lagrer private nøkler og sertifikat i en såkalt *keystore*. Den standarde keytool-implementasjonen implementerer keystore som en fil, og lagrer private nøkler i et passordbeskyttet format for å forhindre uautorisert aksess.

Hver keytool-kommando har en *-keystore* opsjon for å spesifisere navnet og lokasjonen til keystore-filen for keystore'en som håndteres av keytool. Som standard lagres keystore-filen i en fil med endelsen *.keystore* i brukerens hjemmekatalog. Alle keystore-forekomster (nøkler og sertifikater) aksesseres via unike *aliaser*. Aliaser er ikke følsom for store og små bokstaver, slik at aliaset *Lalias* og *lalias* vil referere til samme keystore-forekomst. En alias spesifiseres når man legger til en entitet til en keystore, ved bruk av *-genkey* kommandoen, som genererer et privat/offentlig nøkkelpar. Etterfølgende keytool-kommandoer må bruke dette aliaset for å referere til entiteten.

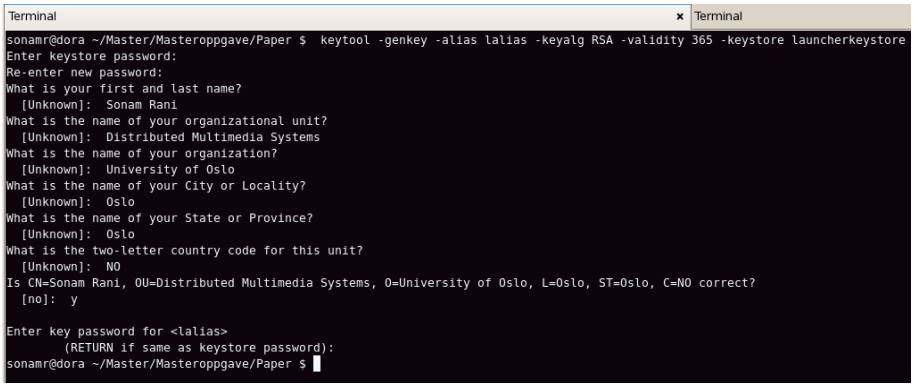
Keytool tillater utviklere å spesifisere algoritmen som skal brukes til nøkkelgenereringen og signaturen, og angis i henholdsvis *-keyalg* og *sigalg* opsjonene. Som standard brukes nøkkelgenereringsalgoritmen, *Digital Signature Algorithm (DSA)*, men *Rivest, Shamir and Adleman (RSA)*-algoritmen støttes også. Signaturalgoritmen deriveres fra algoritmen brukt til nøkkelgenereringen, så hvis den private nøkkelen er av typen DSA, så vil signaturalgoritmen være *SHA1* med DSA, og dersom den private nøkkelen er av typen RSA så vil signaturalgoritmen være *MD5* med RSA. Keytool tillater også utviklere å spesifisere hvor mange dager sertifikatet skal anses som gyldig, med opsjonen *-validity*.

Jeg bruker keytool med aliaset *lalias* for å generere et nøkkelpar. Jeg velger nøkkelgenereringsalgoritmen, RSA, og at gyldigheten skal settes til 365 dager. Jeg oppretter en fil i samme mappen som heter *launcherkeystore*,

som er keystore-filen som lagrer den private nøkkelen som blir generert:

```
keytool -genkey -alias lalias -keyalg RSA -validity 365 -keystore launcherkeystore
```

Etter å ha tastet inn keytool-kommandoen i et kommandovindu, blir utvikleren presentert med informasjon som skal fylles ut om eieren av nøklene og det assosierte sertifikatet. Keytool produserer *X.509 sertifikater*. Først og fremst bes det om et passord til keystore-filen, for å beskytte den private nøkkelen. Deretter bes det om informasjon om utvikleren:



Figur 10.2: Generering av offentlig/privat nøkkelpar og assosiert sertifikat, med Keytool.

For å eksportere sertifikatet lagret i keystore i en egen fil, tastes kommandoen vist nedenfor. Filnavnet sertifikatet skal lagres under, angis i opsjonen *-file*. Jeg har valgt å lagre sertifikatet i filen, *AppLauncherCert.cer*.

```
keytool -export -alias lalias -file AppLauncherCert.cer -keystore launcherkeystore
```

For mer om Keytool, se [34, 38]

Etter å ha generert nøkkelparet og det assosierte sertifikatet med keytool, kan den offentlige nøkkelen distribueres, og den private nøkkelen kan brukes til å generere den digitale signaturen. Applets kan inneholde mange filer og kilder som den bruker til å fremvise sitt innhold, som kan lagres i separerte filer. *JAR*-funksjonen tillater pakking av .class-filer, bilder, lyder og annen digital data i en enkel fil, for rask og enkel distribusjon. Jar-kommandoen lager et såkalt *Java Archive* med all innholdet, og en slik jar-fil kan produseres fra kommandolinjen. For å samle alle kildene jeg bruker til applet'en, lager jeg et slikt arkiv, kalt *AppLaucnher.jar*:

```
jar cvf AppLauncher.jar *.class *.gif
```

På grunnlag av .jar-filen og den private nøkkelen, kan den digitale signaturen determineres. For å signere sertifikatet assosiert med de genererte nøklene med den digitale signaturen, benytter jeg *Jarsigner*. Jarsigner er et verktøy som brukes til å signere jar-filer, og å verifisere signaturer og

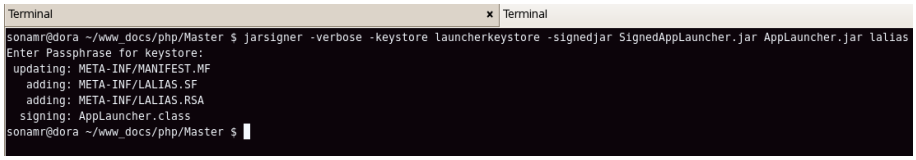
integritet av en signert jar-fil. Jarsigner bruker nøkkel- og sertifikatinformasjon fra en keystore til å generere digitale signaturer for en jar-fil. Etter at den digitale signaturen anvendes på en jar-fil, vil den signerte jar-filen inneholder blant annet en kopi av sertifikatet fra keystore for den offentlige nøkkelen korresponderende til den private nøkkelen brukt til å signere denne jar-filen. Jarsigner kan verifisere den digitale signaturen av den signerte jar-filen ved å bruke kopien av dette sertifikatet, som befinner seg inni den signerte jar-filen [37].

Når man bruker jarsigner til å signere en jar-fil, må man spesifisere aliaset for keystore-forekomsten som inneholder den private nøkkelen som trengs for å generere signaturen. Denne angis bak navnet på jar-filen, altså <*jar-fil*> <*alias for privat nøkkel i keystore*>, i en jarsigner kommando. Jarsigner har en opsjon *-keystore* for å spesifisere keystore-filen som skal brukes. En slik keystore er påkrevd når man signerer en jar-fil, på grunn av den private nøkkelen som behøves til signeringen. Opsjonen *-signedjar* spesifiserer navnet som skal brukes på den signerte jar-filen. Dersom ingen navn er spesifisert i kommandoen, så brukes navnet til jar-filen som skal signeres, altså den opprinnelige jar-filen overskrives av den signerte jar-filen. Hvis *-verbose*-opsjonen er med i kommandoen, så vil jarsigner gi ut ekstra informasjon til utvikleren om signeringen eller verifiseringen. *-storepass* og *-keypass* opsjonene inneholder passordet som kreves til å aksessere henholdsvis keystore-filen, og passordet som brukes til å beskytte den private nøkkelen i keystore-forekomsten, som refereres til av aliaset som er spesifisert i kommandolinjen. Begge passordene kreves kun til signering av jar-filer, og ikke verifisering. Dersom ingen storepass-passordet ikke er oppgitt, så blir utvikleren forespurt om å oppgi det. Dersom keypass-passordet ikke er oppgitt og ulikt storepass-passordet, så foreprørres også utvikleren om å oppgi det [37].

Jeg ønsker å signere jar-filen vi lagde av appletens kilder, AppLaucnher.jar. Jeg ønsker ikke å overskrive den opprinnelige jar-filen, så jeg bruker signedjar-opsjonen til å angi at den signerte jar-filen skal hete *SignedAppLauncher.jar*. Jeg ønsker i tillegg å se informasjon om signeringen og inkluderer derfor verbose-opsjonen i kommandoen, og angir keystore-filen til å være launcherkeystore, og lalias som er aliaset til keystore-forekomsten:

```
jarsigner -verbose -keystore launcherkeystore -signedjar SignedAppLauncher.jar AppLauncher.jar lalias
```

Siden jeg ikke angir noe storepass-passord, så blir jeg spurt om det. Keypass-passordet er i dette tilfellet det samme, og jeg trenger dermed kun å oppgi passordet en gang:



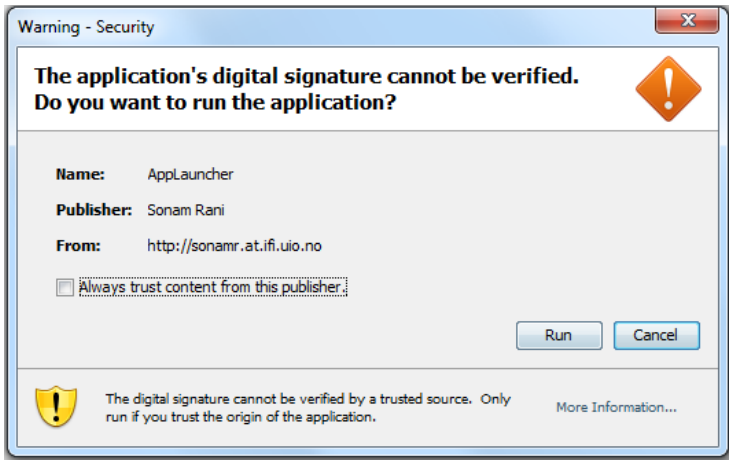
Figur 10.3: Signering av appletens jar-filen, med Jarsigner.

For mer om Jarsigner, se *94, 97*.

Ettersom at applet-elementet tillater å inkludere ressurser til appleten i archieve-attributtet, inkluderer jeg den signerte jar-filen i appleten. På denne måten vil kildene lastes inn før appleten eksekverer, som også vil gjøre at applet'en eksekverer raskere:

```
<P><APPLET code="AppLauncher.class" archive="SignedAppLauncher.jar" width="600" height="400">
  <param name="numberOfFriends" value="<?php echo $counter ?>"/>
  <?php for ($i=0; $i<$counter; $i++) {?>
  <param name="display<?php echo $i; ?>"
  value="Place a call to <?php echo $friends[$i]; echo '( ' . $friends_adrs[$i] . ' )'; ?>"/>
  <?php } ?>
</APPLET></P>
```

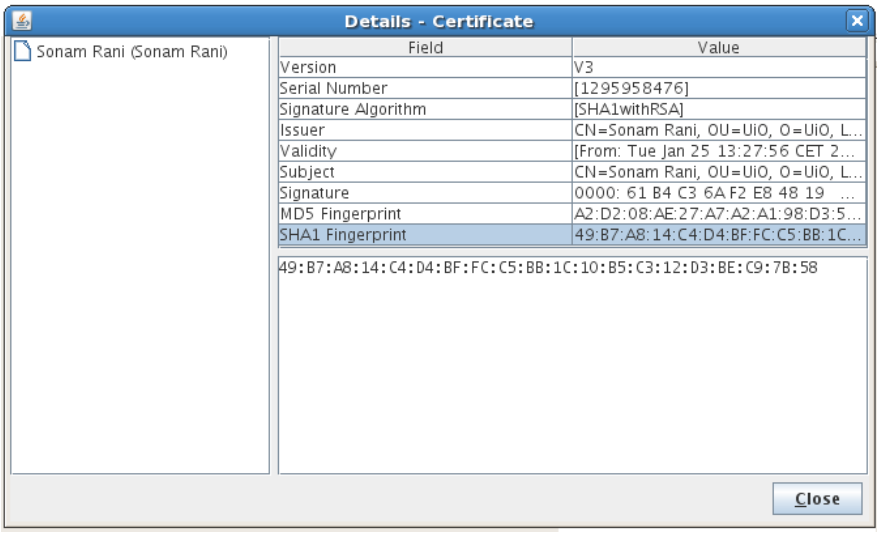
Siden applet'en jeg har lagt er en selvsignert applet, vil brukere få en advarsel om dette når de besøker applet-siden i VoIPBook-applikasjonen. Brukeren kan dermed på egenhånd bestemme om han ønsker å la applet'en eksekvere. Advarselen inneholder navnet på sertifikatet, og navnet på utgiveren av sertifikatet, i tillegg til URL'en til siden som distribuerer applet'en som sertifikatet er for:



Figur 10.4: VoIPBook: Advarsel om selvsignert applet i applet-siden.

Brukeren kan trykke på knappen *more information* i advarselen, som gir detaljert informasjon om det selvsignerte sertifikatet. Her kommer blant annet opp sertifikatets *Version*, som angir hvilken av X.509 standardene som

anvendes på sertifikatet. Videre så vises sertifikatets *Serial Number*. Entiteten som opprettet sertifikatet er ansvarlig for å tildele sertifikatet et entydig serienummer for å skille sertifikatet fra andre sertifikater entiteten oppretter. Dersom et sertifikat oppheves, så vil dette serienummeret legges til en sertifikatopphevingsliste. Feltet *Signature Algorithm* identifiserer hvilken algoritme som ble brukt til å signere sertifikatet. *Issuer* angir navnet til entiteten som har signert sertifikatet. Dette er som regel enten en sertifiseringsautoritet, eller utvikleren av appleten selv dersom sertifikatet er selvsignert. Ved å bruke godkjenne sertifikatet impliserer tillit til entiteten som signerte sertifikatet. Sertifikatets gyldighetsperiode angis med en startdato og en sluttdato, og finnes i feltet *Validity*. Et sertifikat kan vare i alt fra noen sekunder, til et århundre, og avhenger av faktorer som: styrken på den private nøkkelen eller hvor mye man er villig til å betale for sertifikatet. Dette er den forventede perioden brukere kan stole på den offentlige nøkkelen, gitt at den private nøkkelen ikke er blitt avslørt. Feltet *Subject* er navnet til entiteten som eier den offentlige nøkkelen som sertifikatet identifiserer. Her vises blant annet entitetens navn, organisasjonsenhet, og land. Sertifikatet viser også den digitale signaturen som er anvendt på det signerte sertifikatet, i bits, under *Digital Signature*. Det viser i tillegg sertifikatets *Fingerprints* i bytes [2] [37]:



Figur 10.5: Detaljer om det selvsignerte sertifikatet.

[2]Fingerprints (public key fingerprints) er en kort sekvens av bytes som brukes til å autentisere eller slå opp en lenger offentlig nøkkel. Fingerprints lages ved å anvende kryptografiske hashfunkjsoner på offentlige nøkler. Ettersom fingerprints er kortere enn den offentlige nøkkelen den refererer til, kan visse nøkkelhåndteringsoppgaver forenkles.

Dersom brukeren godkjenner sertifikatet, så får applet'en aksess til brukerens maskin, og kan begynne å eksekvere koden den inneholder.

10.3 Applet-koden

Innholdet til Java-applet'en i VoIPBook-applikasjonen, er skrevet i klassen *AppLauncher* i filen *AppLauncher.java*, som ligger på samme filområde som filene til VoIPBook-applikasjonen. AppLauncher-klassen er subklasse av *java.applet.Applet*. Denne Applet-klassen er en subklasse av klassen, *java.awt.Panel*, som i sin tur er subklasse av *java.awt.Container* etter klassen *java.awt.Component*, som igjen er subklasse av *java.lang.Object*.

Applet-klassen har fire klassemetoder, *init()*, *start()*, *stop()* og *destroy()* [1]. Disse metodene kalles automatisk av nettleseren men er initielt tomme metoder, og kan derfor overskrives av utvikleren for å utføre bestemte oppgaver under. Init-metoden kalles på av nettleseren før eksekveringen av applet-koden, for å informere applet'en om at den har blitt lastet inn i nettsiden. Ettersom metoden kalles før eksekveringen, så kan den brukes til initialiseringsoppgaver.

I AppLauncher-klassen oppretter jeg to arrayer, *friends* og *buttons*. Friends-arrayet skal inneholde kontaktinformasjon om applikasjonsbrukerens venner, som sendes fra applet-elementets param-attributter. Hver slik venn skal få en knapp som viser vennens navn og kontaktadresse, og skal fremvises i applet'en for applikasjonsbrukeren. Buttons-arrayet skal inneholde disse knappene, arrayet er av typen *JButton*, fra Java-pakken *javax.swing*. Jeg har opplevd problemer med å fremvise JButtons i nettlesere på Mac-plattformen, da swing-pakken ikke alltid finnes i Java-plugin'en for nettlesere i et Mac-miljø. Derfor holder jeg et array til, *mac_buttons*, som inneholder knapper av typen *Button*, og som brukes dersom brukeren befinner seg på en Mac-plattform.

Applet-klassen arver metoden *setLayout()* fra Container-klassen, som setter visningstypen til applet-panelet knappene skal fremvises på. I AppLauncer bruker jeg en *FlowLayout*, som typisk brukes til å ordne knapper i et panel. Applet-klassen arver også metoden *setForeground()* fra Component-klassen, som angir forgrunnsfargen til applet-komponentene i panelet [1]. I subklassen setter jeg forgrunnsfargen til å være svart. Både friends- og buttons-arrayet, og visningsutseendet intialiseres i init-metoden:

```
friends = new ArrayList<String>();
buttons = new ArrayList<JButton>();
setLayout(new FlowLayout());
setForeground(Color.BLACK);
```

Etter at applet'en har blitt lastet inn av nettleseren og init-metoden har blitt kalt, så kaller nettleseren på metoden start(), for å informere appleten om at eksekveringen kan begynne. I likhet med init-metoden så er start-metoden tom, og kan overskrives av utvikleren i subklassen. I denne metoden henter jeg ut informasjon fra applet-elementets parameterattributter,

fyller arrayene, og lager knappene som skal vises i panelet. Parametrene som sendes i applet-elementet, kan hentes ut ved bruk av Applet-klassens klassemetoden *getParameter()*. Metoden tar i mot en string som argument for å identifisere parameteren vi ønsker å hente verdi fra. Denne stringen er typisk verdien i name-variabelen i param-attributtet. Den tilhørende verdien i value-variabelen returneres av metodekallet.

Den første parameteren er antall venner som har oppgitt en SIP-adresse, og heter *numberOfFriends*. Denne verdien antyder også hvor mange parametere som følger i applet-elementet. Videre går jeg i en for-løkke i like mange iterasjoner som numberOfFriends-verdien, og henter ut navnet og adressen til hver av applikasjonsbrukerens venner. Verdien er til parameternavnet *display+i*, hvor telleren *i* går fra 0 til verdien av numberOfFriends. Etter å ha hentet verdiene i parameter-attributtene, legger vi dem i friends-arrayet:

```
int numFriends = Integer.parseInt(this.getParameter("numberOfFriends"));

for (int i=0;i<numFriends;i++) {
   String friend = this.getParameter("display"+i);
   friends.add(friend);
...
}
```

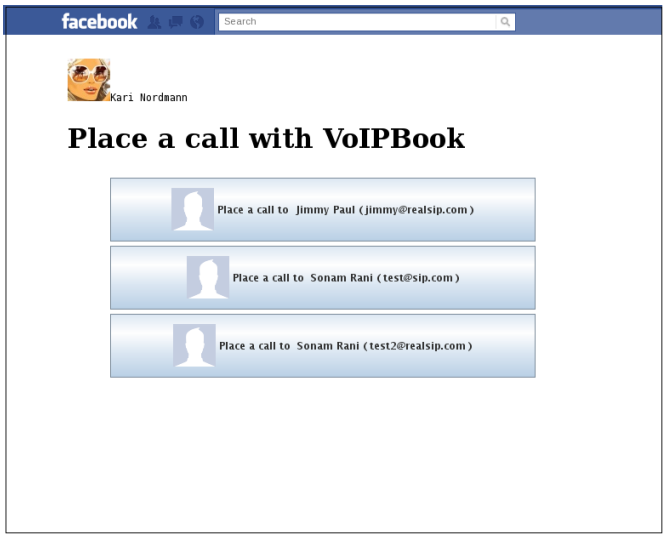
For hver venn i friends-arrayet, lager vi så en knapp. Teksten på knappen får verdien av stringen som ligger i arrayet, altså vennens navn og adresse. Hver knapp får også et bilde som viser et Facebook-hode. Knappene utstyres med en *addActionListener* som lytter etter hendelser som involverer knappen, for eksempel hvis knappen trykkes på av applikasjonsbrukeren. Etter at knappen har blitt deklarert og initialisert, så legges den til å knappe-arrayene:

```
for (int i=0;i<numFriends;i++) {
   String friend = this.getParameter("display"+i);
   friends.add(friend);

   ImageIcon icon = new ImageIcon("pr_image.gif");
   JButton jb = new JButton(friend, icon);
   jb.setPreferredSize(new Dimension(500, 75));
   jb.setActionCommand(friend);
   jb.addActionListener(this);
   buttons.add(jb);
   mac_buttons.add(b);
}
```

Helt til slutt i start-metoden, så legges knappene til i panelet, for fremvisning i appleten:

```
for (int i=0;i<buttons.size();i++) {
    add((JButton)buttons.get(i));
}
```

Fremvisningen til applet-siden i VoIPBook-applikasjonen vil dermed være slik, merk at knappene vil se litt annerledes ut i en nettleser i et Mac-miljø:



Figur 10.6: VoIPBook: Fremvisning av appleten.

Alle knappene har en hendelseslytter. Dette gjør at når en bruker trykker på en knappen, så vil metoden *actionPerformed()* bli kalt, med hendelsesobjektet som argument til metoden. I denne actionPerformed-metoden kan vi definere hva som skal gjøres når en hendelse inntreffer, og kan være individuelt for hver knapp. For å finne ut hvilket objekt som hadde en hendelse, kan vi hente ut stringen som assosieres med hendelsesobjektet ved å kalle metoden, *getActionCommand()*. Denne stringen vil være etiketten til knappen som ble trykket på i appleten:

```
public void actionPerformed(ActionEvent evt) {
    String label = evt.getActionCommand();

      ...
}
```

Som vi ser av figuren ovenfor, så vil alle knappene ha etiketter med teksten: *Place a call to <navn> (<adresse>)*. Applikasjonsbrukeren kan plassere en samtale mot ønsket venn, ved å trykke på knappen med vennens navn og adressen han ønsker å benytte i sesjonen. Teksten på knappene er slik for at brukeren enkelt skal se hvem han kan ringe, da det ikke alltid går frem av adressen alene. Imidlertid så har vi ikke behov for navnet til mottakeren når vi skal starte samtalen, vi trenger kun adressen. For å hente ut adressen, har jeg lagd en metode *parse()*, som tar knappens etikett som argument, og returnerer kun adressen i stringen, og legger til prefikset *sip:*.:

```
public void actionPerformed(ActionEvent evt) {
    String label = evt.getActionCommand();
    String callee = parse(label);

      ...
}
public String parse (String s) {
    String tmp = "";
    for (int i=0; i<s.length(); i++) {
   if (s.charAt(i) == '(') {
           while (s.charAt(++i) != ')') {
  tmp += s.charAt(i);
    }
```

```
 }
     }
    String trimmed = tmp.trim();
    String prefixed = "sip:"+trimmed;
    return prefixed;
}
```

Etter å ha parsert etiketten til knappen som ble trukket på og SIP-adressen til vennen som applikasjonsbrukeren ønsker å ringe er hentet ut, så kalles metoden *launch()*. Denne metoden tar vennens SIP-adresse som argument. Launch-metoden skal foreta åpningen av SJPhone-klienten, og plassere et anrop med adressen den tar i mot som argument.

Ettersom applikasjonsbrukeren kan befinne seg på forskjellige plattformer og plattformene har forskjellige filsystemstrukturer, så må åpningen av SIP-klienten gjøres etter hvilken plattform applikasjonsbrukeren benytter VoIPBook fra. For å determinere hvilket operativsystem brukeren sitter på, kaller jeg på systemmetoden *getProperty()*. Metoden tar en string-nøkkel som argument, og returnerer systemegenskapene til den angitte nøkklen, i form av en string. Jeg kaller metoden med parameteren *os.name*:

```
public void launch (String callee) {
    String osType = System.getProperty("os.name");
       ...
   }
```

I applikasjonen hensyntar jeg kun tre plattformer, nemlig hvis brukeren befinner seg på en *Windows*, *Linux* eller *Mac OS X*-plattform. osType-variabelen vil være på formen *Windows <versjon>* dersom brukeren er i et Windows-miljø. Isåfall kaller jeg på en metode *windows*, som tar seg av åpning av SJPhone i henhold til Windows sin filsystemstruktur. Dersom brukeren er i et Linux-miljø eller Mac-miljø, kaller jeg henholdsvis metodene *linux* og *mac*, som åpner klienten i henhold til miljøets filsystemstrukturen. Alle de tre operativsystem-spesifikke metodene tar mottakeradressen som parameter:

```
public void launch (String callee) {
    String osType = System.getProperty("os.name");
    if (osType.startsWith("Windows"))  windows(callee);
    else if (osType.startsWith("Linux")) linux(callee);
    else if (osType.startsWith("Mac OS X")) macOSX(callee);
    else {
      appErr = true;
      repaint();
    }
}
```

Dersom brukeren sitter på et annet operativsystem enn disse tre, så vil det skrives ut en feilmelding til brukeren. Applet-klassen har en metode *paint()* som den arver fra Container-klassen. Metoden tillater utviklere å skrive beskjeder til brukeren på applet-panelet, og blir kalt automatisk av nettleseren. Dersom utvikleren ønsker å skrive ut en ny beskjed i panelet, så kan man kalle på paint-metoden på nytt, med kallet *repaint()*. Denne metoden er praktisk for å gi brukeren feilmeldinger:

```
 public void paint(Graphics g) {
if (appErr) {

    g.drawString("Could not place call. Make sure SJPhone is installed correctly.
           This applet can only be run on Windows, Linux or Mac OS X platforms.",100,100);
}
    }
```

For en fullstendig oversikt over applet-koden, se filen *AppLauncher.java* under vedlegg A.

10.4 Applet over Windows-plattform

Dersom man laster ned og installerer en applikasjon i Windows-plattformen, vil applikasjonens eksekveringsfil og andre datafiler som regel bli lagret under harddisken, *C:*, og katalogen *Program Files*. Når man laster ned SJPhone for Windows, så vil det opprettes en mappe som heter *SJPhone 1.65*, som inneholder eksekveringsfilen, *SJphone.exe*. SJphone-mappen blir da en undermappe av Program Files.

For å flytte seg til programfilkatalogen, kan man bruke den absolutte stien fra harddisken. En absolutt sti kan angis med en «forover-slash» i starten, og stien fra rotkatalogen til den ønskede katalogen. For å flytte seg brukes stien bak *change directory (cd)*-kommandoen:

```
cd C:\Program Files\SJphone 1.65
```

En av *mijøvariablene* i Windows inneholder den absolutte stien til programfilkatalogen, og heter *%PROGRAMFILES%*. Det går dermed også an å benytte denne miljøvariabelen til å navigere seg til katalogen:

```
cd $(PROGRAMFILES)\SJphone 1.65
```

Etter å ha forflyttet seg til rett katalog, så ønsker vi videre kjøre SJPhones eksekveringsfil får å få klienten til å starte. Vi ønsker også å gi med en SIP-adresse, slik at SJPhone initierer en samtale mot adressen i det den starter. SJPhone kan starte et slikt anrop ved oppstart, ved å angi adressen bak eksekveringsfilen, når eksekveringsfilen kjøres. Adressen til vennen som applikasjonsbrukeren ønsker å ringe ligger i parameteren, callee. For å starte SJphone med denne adressen, må vi utføre følgende kommando:

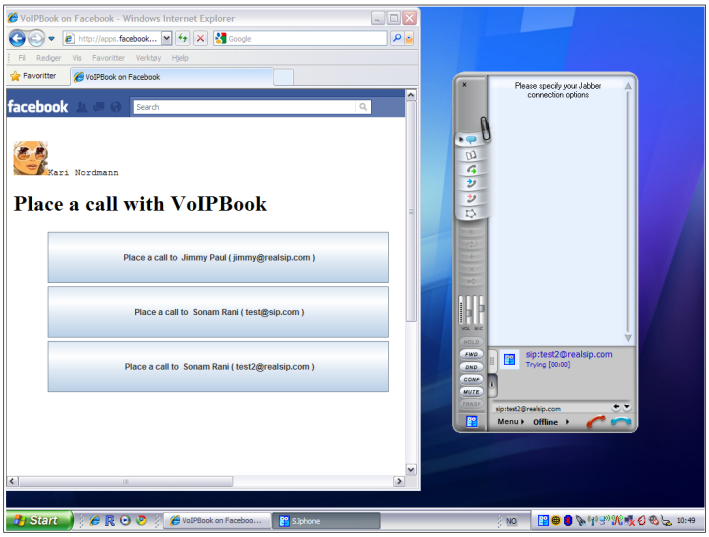
```
SJphone.exe callee
```

For å utføre kommandoen så bruker vi *Java Runtime*-klassens *exec()*-metode. Metoden tar i mot et sett kommandoer og argumenter som skal utføres. Disse kommandoene utføres i en separert subprosess. Arbeidsmappen til subprosessen kan angis som en parameter, og er i dette tilfellet SJPhone 1.65-katalogen, som er en undermappe av programfilkatalogen. Videre tar exec-metoden i mot miljøinnstillinger som skal gjelde når kommandoene utføres. Dersom denne parameteren er satt til null, så arver subprosessen miljøinnstillingene til den nåværende prosessen.

Kallet på exec()-metoden gjøres i en try/catch-blokk, og skriver ut en feilmelding til brukeren dersom kommandoene ikke ble utført, og SJPhone ikke ble åpnet:

```
public void windows (String callee) {

       String programfiles = System.getenv("PROGRAMFILES");
       programfiles += "\\SJphone 1.65";
       String[] cmd = {"SJphone.exe", ""+callee};
try {
    Runtime.getRuntime().exec(cmd, null, new File(programfiles));
}
catch (Exception e) {
    appErr = true;
    System.out.println("\nProblem trying to launch application SJPhone, and calling "+callee+".\n");
    e.printStackTrace();
    repaint();
}
}
```

Slik vil det se ut etter en vellykket åpning av SJPhone fra VoIPBook-applikasjonen i Windows-plattformen, da brukeren har angitt SIP-adresse til vennen han ønsker å ringe ved å trykke på knappen med vennens navn og adresse. I eksemplet ringer Kari Nordmann (som applikasjonsbruker) til Sonam Rani med SIP-adressen, «test2@realsip.com», som Sonam Rani har oppgitt til VoIPBook-applikasjonen:



Figur 10.7: AppLauncher:SJPhone åpnet i Windows-plattformen.

Koden ovenfor er testet og fungerer på en SJPhone-klient i en Windows-plattform. Fordelen med en slik løsning er at applikasjonsbrukere kan starte en VoIP-samtale med vennen han ønsker å ringe, med et klikk i applet-siden av applikasjonen. Navigering til rett katalog, og åpning av SIP-klienten gjøres transparent for brukeren ettersom appleten har fått tilgang til brukerens maskin, og kan utføre kode fra serversiden, på klientsiden.

Et alternativ til løsningen over, er å lage et batch-skript med multiple kommandoer samlet i en fil. I stedet for å kalle exec-metoden med kommandoene som ovenfor, så kan exec-metoden heller kalle skriptet. Dette åpner for muligheten for å benytte Facebook-applikasjoen over flere SIP-klienter. I et slik skript kan man eksempelvis angi hvilken SIP-klient man ønsker å starte opp, som blant annet SJPhone, XLite, Skype med SIP eller en annen klient. Batch-skriptet kan spørre brukeren om hvilken SIP-klient som skal brukes, eller prøve å finne hvilke SIP-klienter som er tilgjengelige på brukerens maskin, ut i fra en prioritert liste over klienter som støttes. Det ene kravet til en slik løsning er at samtlige klienter kan ta en adresse som argument når eksekveringsfilen kjøres, og initiere en sesjon med adressen ved oppstart. Det andre kravet er at SIP-klientene tar denne adressen på

samme format, altså standard SIP-format, som «sip:navn@domene».

Batch-skriptet nedenfor er et eksempel på et skript som tillater applikasjonsbrukere å enten starte SJPhone, eller XLite. Skriptet mottar to argumenter, nemlig hvem av SIP-klientene som skal åpnes og hvilken SIP-adressen klienten skal initiere en sesjon med, ved oppstart.

Skriptet setter variabelen *program* til å holde på programnavnet som mottas som argument, og variabelen *callee* til å holde på adressen. Først så kjøres kodesnutten *check_program*, som avgjør om programmet brukeren ønsker å starte er XLite eller SJPhone. Dersom argumentet inneholder et annet program enn SJPhone eller XLite så hopper skriptet til kodesnutten *failed_program* som gir en feilmelding til brukeren, og avslutter. Det er imidlertid enkelt å utvide et slikt skript til å starte flere enn to SIP-klienter.

Avhengig om brukeren har tastet SJPhone eller XLite så kjøres henholdsvis kodesnuttene *start_sjphone_program* eller *start_xlite_program*. I begge tilfellene så navigeres det til riktig katalog hvor eksekveringsfilen ligger, og kjører eksekveringsfilen med adressen, som ligger i callee-variabelen. Dersom det skulle forekomme en feil, for eksempel at skriptet ikke finner eksekveringsfilen, eller at formatet på adressen er feil, så hopper skriptet til *failed_launching*, og brukeren får beskjed om at det oppstod en feil. Ettersom det er forskjellige feilsituasjoner som angis med forskjellige ikke-null verdier i variabelen *%ERRORLEVEL%*, så sjekker jeg ikke eksplisitt for hver feilsituasjon, men heller om variabelen ikke har verdien 0, som er den feilfrie tilstanden. Dersom alt går som det skal, så hopper skriptet til *end_program* etter å ha startet klienten, og avslutter skriptet:

```
@echo Launching...

set program=%1
set callee=%2
GOTO :check_program

:check_program
if %program% == SJPhone GOTO :start_xlite_program
if %program% == Skype GOTO :start_skype_program
if %ERRORLEVEL% NEQ 0 goto :failed_program
:end

:start_sjphone_program
cd %PROGRAMFILES%
cd SJPhone 1.65
SJphone.exe %callee%
if %ERRORLEVEL% NEQ 0 goto :failed_launching
goto :end_program
:end

:start_xlite_program
cd %PROGRAMFILES%
cd Xlite
Xlite.exe %callee%
if %ERRORLEVEL% NEQ 0 goto :failed_launching
goto :end_program
:end

:failed_program
@echo Program is invalid.
:end

:failed_launching
@echo Error launching program.
:end

:end_program
@pause
:end
```

10.5 Applet over Linux-plattform

I Linux-miljøet så finnes det ingen bestemt mappe hvor programfiler skal ligge, slik som mappen *Program Files* i Windows. Det finnes følgelig heller ingen miljøvariabel som viser til stien til en slik mappe, som variabelen *%PROGRAMFILES%* i Windows. Dette betyr også at brukeren kan plassere eksekveringsfilen til et program, så å si vilkårlig i filsystemet. Imidlertid finnes det kataloger hvor programfiler vanligvis plasseres i Linux filstrukturen.

Rotmappen i Linux angis med en enkel /, som er det øverste leddet i filhierarkiet. Undermappen */usr* er for brukerprogrammer, og inneholder biblioteker, dokumentasjon og kildekode for slike brukerapplikasjoner. Mappen inneholder mesteparten av verktøy og applikasjoner for flere brukere av maskinen. Undermappen */usr/bin* inneholder binære filer for brukerprogrammer og disse finnes som regel også under */bin*-katalogen. Undermappen */usr/sbin* inneholder binære filer for systemadministratorer, og finnes som regel også under */sbin*. Undermappen */usr/local* inneholder brukerprogrammer som en gitt bruker har installert fra en kilde. Dataene er lokale, og er spesifikk for en bruker. Brukerspesifikke programmer kan også befinne seg i brukerens hjemmekatalog, under */home* [72].

For enkelhetsskyld så forutsetter jeg at SJPhone er installert og lagret under brukerens hjemmekatalog. I Linux vil SJphone-programvaren finnes i en mappe, *SJphoneLnx-299a*. I denne mappen finnes et shell-script, *sjphone.sh*, som kaller eksekveringsfilen som ligger i en ytterligere undermappe, *SJphoneLnx-299a/lib*. I motsetning til Windows, så er eksekveringsfilene i Linux i et «*.elf» format, i stedet for «*.exe». Eksekveringsfilen til SJphone-klienten i Linux heter *sjphone.elf*. For å starte opp klienten, trenger vi å kalle på shell-skriptet, som videre kaller på eksekveringsfilen.

Kommandoen for å starte klienten med SIP-adressen som applikasjonsbrukeren har valgt, vil være å kjøre shell-skriptet, sjphone.sh, og gi med SIP-adressen som argument til filen. Den brukervalgte adressen finnes i variabelen, callee:

```
String command[] = {"sjphone.sh", ""+callee};
```

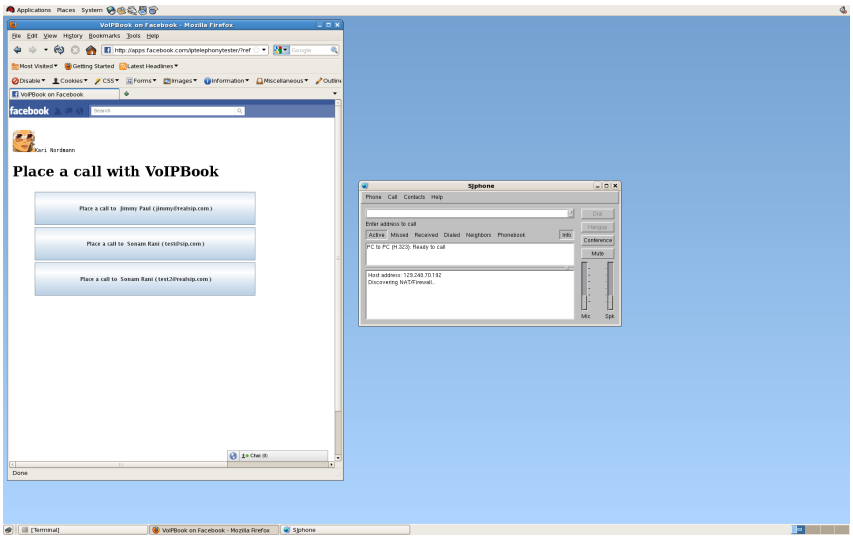
Videre så henter jeg ut stien til brukerens hjemmekatalog hvor vi antar at SJPhone sin mappe med programvare befinner seg, ved å kalle på System-metoden, *getenv()*. Metoden tar i mot en miljøvariabel som parameter i String-format, og returnerer verdien til miljøvariabelen også som en string. Jeg kaller metoden med argumentet, *HOME* for å få den absolutte stien til hjemmekatalogen, og konkatenerer stien med undermappen til SJPhone:

```
String home = System.getenv("HOME");
home += "\SJphoneLnx-299a";
```

For å eksekvere kommandoen, så benyttes Java Runtime-klassens exec-metode. Jeg gir med kommandoen og stien til SJPhone-mappen som arbeidsmappen prosessen skal kjøre i, som argumenter til metoden:

```
 public void linux (String callee) {

String command[] = {"sjphone.sh", ""+callee};
String home = System.getenv("HOME");
        home += "\SJphoneLnx-299a";
try {
    Runtime.getRuntime().exec(command, null, new File(home));
        }
catch (Exception e) {
            appErr = true;
    System.out.println("\nProblem trying to launch application SJPhone, and calling "+callee+".\n");;
    e.printStackTrace();
    repaint();
}
    }
```

Slik vil det se ut etter en vellykket åpning av SJPhone fra VoIPBook i Linux-plattformen:



Figur 10.8: AppLauncher:SJPhone åpnet i Linux-plattform.

Selv om SJPhone er en kryssplattformkompatibel klient, så finnes det flest funksjonaliteter for SJPhone i Windows-miljøet enn i andre miljøer. I en SJPhone-klient i Linux-plattformen, så støtter programvaren ikke å gi med argumenter til klienten ved oppstart. Dette gjør at applikasjonsbrukere ikke får starten klienten med en ønsket SIP-adresse, i Linux. Som vi ser av bildet ovenfor så vil klienten imidlertid starte, men brukeren må selv taste inn SIP-adressen til vennen han ønsker å ringe, etter at klienten åpnes. På denne måten mistes litt av «klikk-og-ring»-effekten til VoIPBook, men brukeren vil fremdeles få en oversikt av applikasjonen over venners SIP-adresser, som han kan taste inn i klienten.

I koden ovenfor er forutsetningen for at brukeren skal kunne benytte VoIPBook-applikasjonen med SJPhone i et Linux-miljø, at SJPhone-mappen befinner seg i hjemmekatalogen til brukeren. En løsning som gjør det mulig å tillate applikasjonsbrukere å starte SJPhone uavhengig av hvor map-

pen med programvaren er, er å lage et skript som lokaliserer SJPhone sin programvaremappe som inneholder skriptet som kjører eksekveringsfil, og returnerer stien til mappen. Et slikt skript vil også åpne for at flere SIP-klienter kan benyttes. Dersom skriptet ikke finner stien til en slik mappe, kan skriptet videre søke etter mappe med programvare for en annen SIP-klient, eksempelvis XLite. Skriptet kan prioritere hvilken SIP-klient som skal brukes, ved å utføre søket i en prioritert rekkefølge. Ettersom at SJPhone ikke tillater å plassere et anrop til en adresse ved oppstart i Linux-miljøet, så kan man prioritere andre klienter som tillater det.

Et eksempel på et slikt skript, er vist nedenfor. Skriptet forsøker i finne lokasjonen til mappe med programvare for SJPhone og XLite. I metoden, *check_program()*, så sjekkes det om lokasjonene er tomme eller inneholder en sti. Dersom det finnes en sti, så kalles metodene *sjphone_start()* eller *xlite_start()* som henholdsvis starte SJPhone eller XLite med adressen, som gis til skriptet i variabelen, callee. Dersom ingen av klientene finnes på brukerens maskin, og begge stiene dermed er tomme, så kalles metoden *fault_ending()* som skriver en feilmelding til brukeren, og terminerer skriptet:

```
#!/bin/bash

function end_program() {
echo "Ending program."
exit 1
}
function sjphone_start() {
echo "Loading Sjphone.."
cd $sjphone_loc
sh sjphone $callee
end_program
}
function xlite_start() {
echo "Loading Xlite.."
cd $xlite_loc
sh xlite $callee
end_program
}
function check_program() {
    if [ "$sjphone_loc" != "" ]; then
sjphone_start
    elif ["$xlite_loc" != "" ];then
xlite_start
    else [ fault_ending ]
    fi
}
function fault_ending() {
    echo "Found no such SIP-clients."
    kill $$
}

callee=$1
sjphone_loc=$(find / -name SJphoneLnx-299a)
xlite_loc=$(find / -name Xlite)
check_program
```

Selv om skriptet åpner for at applikasjonsbrukere i Linux-plattformen kan ha SIP-klientens programvare lagret på et vilkårlig sted på maskinen og at applikasjonsbrukere kan benytte applikasjonen med flere SIP-klienter, så finnes det også noen ulemper med skriptet. Ettersom at skriptet må gjøre et søk i hele filsystemet for å lokalisere SIP-klientens mappe, så kan det ta tid før søket er fullført. Dersom man i tillegg skal gjøre et slikt søk for flere klienter, så vil det ta ytterligere tid. Skriptet utføres «bak kulissene», så applikasjonsbrukeren blir sittende å vente denne tiden. I en kjøring med skriptet

ovenfor begrenset jeg søket til å kun sjekke i undermapper av hjemmekatalogen til brukeren, og det tok 1 minutt og 8 sekunder før skriptet fant mappene, og åpnet SJPhone-klienten.

Det er altså fordeler og ulemper med begge løsningene, og utviklere må ta en vurdering av hva som er mest kritisk for brukeropplevelsen: Å forlange at brukeren har lagret programvaren på en bestemt plass, men starte klienten raskt for brukeren. Eller å gi brukere rom til å lagre programvaren hvor de ønsker, men brukeren må vente på at klienten skal lokaliseres og starte opp.

10.6 Applet over Mac OS X-plattform

Roten i OS X filsystemet refereres til med / som i Linux og alle andre Unix baserte operativsystemer, og er foreldermappen til alle andre mapper. */Users*-undermappen inneholder alle brukerkontoer på maskinen med deres tilhørende unike filer og innstillinger, veldig likt */home*-katalogen i Linux. */Library* mappen inneholder delte biblioteker med filer nødvendig for operativsystemet til å fungere riktig, det finnes også en slik mappe under brukermappene, med filer spesifikk for den gitte brukeren. Videre finnes blant annet undermappene */usr*, */bin* og */sbin* som holder på samme filer i Mac OS X, som de gjør i Linux. Mac OS X har i tillegg en egen undermappe under roten for alle applikasjoner som finnes på Mac'en, */Applications*. Det finnes også en slik undermappe under brukermappen, med applikasjoner spesifikk for den gitte brukeren. Sjekk [6] for mer om Mac OS X filstruktur.

SJPhone-klienten i et Mac-miljø vil altså bli lagret i Applications-mappen under brukermappen. For å finne stien til brukermappen, bruker vi miljøvariabelen, *HOME* ved å kalle på systemmetoden *System.getenv()*, med miljøvariabelen som argument. Videre så legger vi til applikasjonsmappen som en del av stien:

```
String home = System.getenv("HOME");
home += "/Applications";
```

I applikasjonsmappen så ligger applikasjonene på formen *ApplikasjonsNavn .app*. Filekstensjonen *.app* betyr at innholdet er en *Application Bundle* (applikasjonspakke), som er en mappe med binære filer og støttende filer til applikasjonen. Eksekveringsfiler i Mac OS X er som regel på formatet *Mach-O*, og ligger i applikasjonspakken. Sjekk [47] for mer om slike applikasjonspakker.

For å starte en applikasjon i OS X, kan man kalle applikasjonspakken med kommandoen *open*, og gi med argumenter til eksekveringsfilen på slutten kommandoen. For å åpne SJPhone med brukerangitt adresse, vil kommando-arrayet dermed være slik, da SJphone sin applikasjonspakke heter *SJphone.app*:

```
String cmd[] = {"open", "SJphone.app", ""+callee};
```

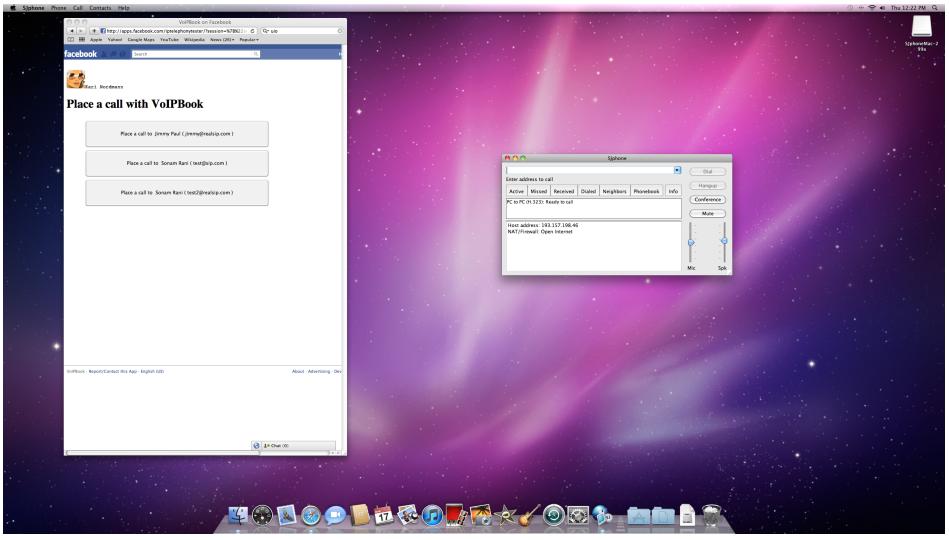
For å utføre kommandoen, kalles Java Runtime sin exec()-metode, med kommando-arrayet og i stien til prosessens arbeidsmappe:

```
Runtime.getRuntime().exec(cmd, null, new File(home));
```

Hele metoden ser slik ut:

```
public void macOSX (String callee) {

String cmd[] = {"open", "SJphone.app", ""+callee};
String home = System.getenv("HOME");
home += "/Applications";
System.out.println(home);

try {
    Runtime.getRuntime().exec(cmd, null, new File(home));
}
catch (Exception e) {
            appErr = true;
    System.out.println("\nProblem trying to launch application SJPhone, and calling "+callee+".\n");;
    e.printStackTrace();
            repaint();
}
```

Slik vil det se ut etter en vellykket åpning av SJPhone fra VoIPBook i Mac-plattformen:



Figur 10.9: AppLauncher:SJPhone startet i Mac OS X-plattform.

Også i Mac-miljøet mangler SJPhone støtte for å starte en sesjon med en adresse, ved åpning av klienten. Dermed kan vi skrive et lignende skript som for Linux-plattformen, som støtter åpning av flere SIP-klienter og kan prioritere klienter som støtter dette. Fordelen her er at vi vet at brukerprogrammer lagres i Applications-mappen, så vi slipper det tungvinne søket som gjør at brukeren må vente.

Nedenfor er et eksempel på et slikt skript som tillater applikasjonsbrukere å benytte XLite, dersom SJPhone ikke er installert. Skriptet definerer

lokasjonen til SJPhone og XLite, og kaller metoden *check_program()*. I denne metoden prøver skriptet først å åpne SJPhone med den brukervalgte adressen som er gitt til skriptet i argumentet, callee. Dersom åpningen var vellykket, kalles metoden *end_program()* som avslutter skriptet. Dersom åpningen ikke var vellykket, så prøver skriptet å åpne XLite som videre kaller end_program() og avslutter, dersom åpningen var vellykket. Dersom ingen av klientene hadde en vellykket åpning, så kalles metoden *fault_ending()* som skriver en feilmelding til brukeren og terminerer skriptet:

```
#!/bin/bash

function end_program() {
        echo "Ending program."
        exit 1
}
function fault_ending() {
        echo "No such SIP-clients."
kill $$
}
function sjphone_start() {
echo "Loading SJPhone.."
open $sjphone_loc "$callee"
}
function xlite_start() {
echo "Loading Xlite.."
open $xlite_loc "$callee"
}
function check_program() {
        sjphone_start()
        if [ "$?"-e 0]; then end_program(); fi
        xlite_start()
        if [ "$?"-e 0]; then end_program(); fi
        fault_ending()
}

callee=$1
home=$HOME/Applications
sjphone_loc=$home/SJPhone.app
xlite_loc=$home/XLite.app
check_program()
```

Kapittel 11

Konklusjon

11.1 Resultater og funn

I oppgaven har vi ønsket å se på hvordan integrasjon med Facebook kan gi økt nettverkseffekt for VoIP-tjenester. Nettverkseffekt innebærer at som et resultat av at flere mennesker benytter en tjeneste, så får tjenesten økt verdi for eiere av tjenesten. Det vi ønsker å oppnå, er altså at flere mennesker skal bruke VoIP-tjenester, slik at VoIP-tjenester får økt verdi for brukere av tjenesten, og kan bli en standard løsning for telefoni.

For å oppnå økt nettverkseffekt for VoIP-tjenester, har vi sett på hva som må til for at flere brukere kan benytte tjenesten. Vi har sett at VoIP-tjenester mangler et offentlig og stort oppslagsverk som tillater VoIP-brukere å slå opp og få kjennskap til andres kontaktinformasjon, som med SIP-adresser. VoIP-tjenester trenger også å tilgjengeliggjøres mer for brukere. Å ringe med tradisjonell telefoni i dag er også veldig «enkelt», da brukere kan slå opp numre, klikke og ringe. En slik «klikk og ring»-funksjonalitet behøves også med VoIP, for at brukere skal oppleve det som enkelt å ringe med VoIP, og dermed kan foretrekke å ringe på en slik måte.

For at VoIP skal oppnå økt nettverkseffekt for tjenesten og potensielt bli en standard telefoniløsning, trengs det altså et større kontaktoppslagsverk for tjenesten, og muligheten til å ringe med VoIP må tilgjengeliggjøres for brukere.

Facebook appellerer ikke bare til forbrukere, men nettstedet tilbyr også tredjepartsutviklere å integrere med Facebook-plattformen. Tredjepartsutviklere kan utvikle applikasjoner for tjenester i Facebook-plattformen, og også utvikle mobilapplikasjoner eller tredjepartsnettsider som integrerer med Facebook-plattformen. På denne måten har Facebook oppnådd en tosidet nettverkseffekt, som gjør at Facebook er stadig viktigere for forbrukere på grunn av tjenester som tilbys gjennom tredjeparter, og samtidig så er tredjepartsutviklingen stadig viktigere for utviklerne på grunn av den

store brukermassen Facebook tilbyr. Som et sosialt nettverk på nett, så er Facebook et naturlig sted for å dele innhold som kontaktinformasjon til VoIP-tjenester.

Facebook kan med sin store brukermasse gi økt nettverkseffekt til VoIP-tjenester, og tilbyr muligheter for tredjepartsutviklere til å integrere tjenester i plattformen.

Facebook-data tilgjengelig for utviklere tilbys i en graf-struktur, hvor grafen inneholder objekter og forbindelser mellom disse objektene. Dersom utvikleren lager en applikasjon i Facebook-domenet, kan hun hente ut informasjon fra dette graf-API, via HTTP. Dersom utvikleren ønsker å hente ut store mengder og kompleks data fra APIet, kan hun bruke FQL. Facebook tilbyr også verktøy for programvareutvikling, som PHP- og JavaScript SDK'ene. Disse verktøyene kan lastes inn i siden til utvikleren, og brukes til å utføre forespørseler på vegne av applikasjonsbrukere og mot graf-APIet. Dersom utvikleren ønsker å lage en integrasjon til Facebook-plattformen fra en ekstern nettside, så tilbyr Facebook sosiale plugins. Disse små interaksjonsobjektene kan enkelt plasseres på tredjepartsnettsider ved å implementere XFBML-elementer i siden. Tilsvarende tilbyr Facebook også SDKer for mobilapplikasjoner som integrerer med Facebook-plattformen, slik som Android SDKet og iOS SDKet.

Facebook tilbyr altså en rekke APIer og SDKer for å tillate integrasjon med Facebook-plattformen. Facebook tillater tilgjengeliggjøring av VoIP-tjenester ved å tilby tjenesten via en tredjepartsapplikasjon i Facebook, eller ved å bruke Open Graph Protocol med en tredjepartsnettside som tilbyr VoIP-tjenester. Med en Facebook-applikasjon kan utvikleren også presentere informasjon fra Facebook om brukeren og brukerens venner, slik at man kan opprettholde et kontaktoppslagsverk. Å utvikle en Facebook-applikasjon som tilbyr VoIP-tjenester kan dermed gi økt nettverkseffekt til tjenesten, ettersom VoIP-tjenester behøver et kontaktoppverkslag og å tilgjengeligjøres mer for brukere.

I siste delen av oppgaven presenterte jeg VoIPBook, en prototype på en Facebook-applikasjon som tilbyr VoIP-tjenester. Applikasjonen implementerer en kontaktoppslagstjeneste som opprettholder SIP-adresser, og tilgjengeliggjør VoIP ved å gi brukere muligheten til å ringe med en VoIP via en SIP-klient. I prototypen bruker jeg en SIP-klient, SJPhone, som er åpen, kostnadsfri og kryssplattformkompatibel. For å la brukere ringe med SIP-klienten, er det en forutsetning av klienten er installert på brukermaskinen. VoIPBook-applikasjonen åpner denne klienten på brukermaskin og gir den input fra kontaktoppslagsdelen av applikasjonen, for å sette i gang en samtalesesjon ved oppstart.

Som vi har sett, så finnes det sikkerhetsrestriksjoner som gjør at applikasjonen trenger tillatelse av brukere for å åpne klienten på brukermaskinen.

For å håndtere dette, har vi har sett på løsninger som nettleser-plugins og cookies. Vi fant at ulempen med nettleser-plugins at de er nettleseravhengige. Utvikleren er enten nødt til å påta seg en stor og kontinuerlig jobb, og utvikle slike plugins for alle nettlesere som støtter applikasjonen, eller begrense brukermassen til brukere av den bestemte nettleseren hun har laget en plugin for. I likhet med nettleser-plugins, så kan cookies også være nettleseravhengige og er derfor ikke en ideell løsning i denne implementasjonen. Vi har også sett på løsninger som Client-side-skripter, og Microsofts ActiveX Controls-rammeverk. Ulempen med Client-side skripter er at de har veldig begrenset tilgang til brukerens maskin, av sikkerhetshensyn. Denne begrensningen kan være såpass stor at skriptet nesten ikke får eksekvert innholdet sitt. ActiveX-kontroller har på den annen side, full tilgang til brukerens maskin. Imidlertid så kan slike kontroller kun brukes i Internet Explorer-nettlesere i et Windows-miljø. Vi har sett på applets, mer bestemt, Java-applets. Slike applets kan tillate utviklere å aksessere brukerens maskin, på innvilgning av brukeren. Java-applets kan brukes i nettlesere som har Java-plugin, noe som de fleste nettlesere har. Java-applets kan også kjøre i flere miljøer.

Vi har altså sett at det finnes mange måter tilby VoIP-tjenester gjennom Facebook-plattformen. Siden vi er interessert i økt nettverkseffekt for tjenesten, så er det implisitt at tjenesten må anvende en løsning som tillater størst mulig brukermasse. En Facebook-applikasjonen med Java-applet som åpner VoIP-klienten på brukermaskinen med input fra applikasjonen, er den løsningen av de alternativene vi har sett på i denne oppgaven som tillater flest mulige brukere å benytte VoIP-tjenesten. En slik løsning vil være nettleseruavhengig og plattformuavhengig.

11.2 Personlig erfaring

Denne oppgaven har gitt meg bred innsikt i VoIP-teknologien og dens behov for økt nettverkseffekt. Jeg har lært at for at teknologien skal vokse frem til å bli en standard for telefoni, så må mange brukere benytte tjenesten som igjen må gjøre at flere brukere ønsker å bruke tjenesten. For at forbrukere generelt skal overtales til å benytte en ny tjeneste eller produkt, så må man fokusere på manglene ved tjenesten eller produktet og tilby en løsning for disse. Problemet for mange leverandører av VoIP-løsninger er blant annet det ikke finnes en offentlig brukerdatabase, og at tjenesten ikke er tilgjengelig nok for brukere.

Gjennom oppgaven har jeg også lært at som en konsekvens av positiv nettverkseffekt til en konkurrent i et marked, så er det en korresponderende negativ nettverkseffekt på alle andre konkurrenter i markedet. Dette er også sant for VoIP-tjenester. I oppgaven så har jeg skjønt at VoIP-klienten, Skype, dominerer dagens VoIP-marked. Giganten har oppnådd en massiv

nettverkseffekt, som gjør det vanskelig for andre VoIP-leverandører å tilby sine VoIP-tjenester. En slik dominans har gjort at mange brukere er under oppfatningen av at VoIP betyr Skype. For å gi økt nettverkseffekt til andre VoIP-tjenester, så opplever jeg at det er mer enn en løsning for mangler som skal til. VoIP-leverandører må også rette sine VoIP-tjenester mot en annen målgruppe eller tilby et unikt tilbud forskjellig fra Skype, for å potensielt få økt nettverkseffekt.

Facebook er en plattform som når ut til både forbrukere og utviklere. Som en utvikler, så har jeg opplevd at Facebook sine SDKer og APIer kan være noe dårlig dokumentert. Dette er en liten hindring for utviklere, da man må bruke litt for mye tid på å forstå hvordan verktøyene fungerer. Som en forbruker, så setter jeg pris på tredjepartsapplikasjonene i Facebook. Slike applikasjoner gir forbrukere muligheten til å prøve produkter og tjenester, og jeg har observert at Facebook-brukere er like villige til å teste ut produkter og tjenester. Dersom tjenesten eller produktet som tilbys gjennom applikasjonen er likt av brukere, så er det lett for brukerne å dele denne erfaringen som kan skape økt nettverkseffekt.

Etter mitt arbeid med oppgaven vil jeg anbefale firmaer og selskaper som ønsker å introdusere et produkt eller en tjeneste, om å vurdere en integrasjon med Facebook-plattformen. På denne måten kan tjenesten eller produktet raskt oppnå en positiv nettverkseffekt, dersom brukere opplever det som et bra produkt eller tjeneste. Større internasjonale selskaper som allerede har funnet sin plass i markedet, bør ta en vurdering om de ønsker å integrere med Facebook, med hensyn på de potensielle nye kundene mot arbeidet for å sette seg inn i dokumentasjonen. Det hender at slike store selskaper ønsker å nå ut til en annen målgruppe, og integrerer eksempelvis heller med online profesjonelle nettverk, som LinkedIn. Imidlertid så er det et faktum at de fleste største internasjonale selskaper har en viss form for integrasjon med Facebook. Jeg har sett at dette som regel forekommer med Facebook sine sosiale plugins på selskapets eksterne nettsider. Slike selskaper bruker også å ha en «like»-side i Facebook, som introduserer selskapets virksomhet, og bevisstgjør Facebook-brukere på selskapets eksistens i stedet for å tilby tjenester.

11.3 Beslektet litteratur

Gigantene *Skype* og *Google* har jobbet med å tilgjengeliggjøre VoIP-tjenester, på samme måte som arbeidet i denne oppgaven. Skype har laget en integrasjon med Facebook i sin VoIP-klient, og Google har implementert VoIP-tjenester i sin epost-klient. I de neste avsnittene skal vi ta en nærmere titt på disse.

11.3.1 Facebook på Skype

Høsten 2010 lanserte Skype og Facebook samarbeid om en integrasjon i ny Skype-programvare. Skype versjon 5.0 og nyere for Windows inneholder en Facebook-kategori som lar brukere forbinde Skype-kontoen og Facebook-kontoen sin. På denne måten kan brukere ringe og sende beskjeder til venner fra Facebook, direkte fra Skype. Brukere har også mulighet til å se nyhetsoppdateringer, kommentere og like Facebook-innhold fra Skype.

Skype-applikasjonen henter ut informasjon om brukerens venner fra Facebook. Dersom brukeren ønsker å ringe en Facebook-venn, vil applikasjonen sjekke om vennen også er en Skype-venn i brukerens kontaktliste i Skype-klienten. Dersom dette er tilfelle, vil det settes opp en Skype-til-Skype sesjon. Dersom Facebook-vennen ikke er en venn i Skype-kontaktlisten til brukeren, vil applikasjonen gjøre et oppslag og sjekke om vennen har en Skype-konto. I tilfellet, vil vennes Skype-ID hentes ut, og bli lagt til i brukerens Skype-kontaktliste. Deretter vil det bli satt opp en Skype-til-Skype sesjon. Hvis vennen ikke har noen Skype-konto, kan brukeren sende en beskjed til vennen i Facebook, og forespørre vennen om å opprette en Skype-konto [8]. Skype-applikasjonen tilbyr dermed et slags *Facebook kontaktoppslag*, som lar applikasjonsbrukere oppdage Skype-venner via Facebook. På denne måten får Skype også markedsført sine tjenester via Facebook.

I en sammenligning av Skype sin Facebook-integrasjon og VoIPBook, ser vi at ideene er veldig like. I begge integrasjonene ønskes det å benytte Facebook som et kontaktoppslagsverk. Skype-applikasjonen benytter Facebook til å finne brukerens Skype-venner via navnet til Facebook-venner. Anropsprosidyren gjøres naturligvis med Skype-klienten. Dette betyr at applikasjonsbrukere både må ha en Skype-brukerkonto og en Facebook-brukerkonto for å benytte tjenesten.

VoIPBook er derimot en applikasjon i Facebook som integrerer VoIP-tjenester i Facebook, og gjør at brukere kun trenger en brukerkonto i Facebook for å benytte kontaktoppslagstjenesten. I VoIPBook har jeg også vært opptatt av å tillate flest mulig brukere å benytte tjenesten ettersom den er kryssplattformkompatibel, i motsetning til Skype-applikasjonen som kun tillater Windows-brukere. VoIPBook samler informasjon om venner fra Facebook, men lar brukere og venner oppgi kontaktinformasjon i applikasjonen. VoIPBook tillater også brukere å ringe med VoIP, brukeren må i tilfellet ha installert SJPhone-klienten, eller eventuelt en annen klient.

11.3.2 Google video og voice plugin

I 2008 lanserte Google en nettleser-plugin for voice- og video-chat, *Video and voice plugin*. Ved å laste ned og installere denne nettleser-plugin'en, kan GMail-brukere plassere audio og videosamtaler med andre GMail brukere,

via nettleseren [24].

Når en bruker ønsker å ringe en annen GMail-bruker i GMail, så får brukeren en liste over alle GMail-kontaktene han har. De kontaktene som har installert denne video- og audio-plugin'en vil få et kamera-ikon ved navnet i listen. Brukeren kan starte en video- eller audio-samtale med kontakten ved å klikke på dette ikonet. Dersom en kontakt ikke har installert denne nettleser-plugin'en, så kan brukeren invitere GMail-kontakten til å laste ned og installere nettleser-plugin'en for voice- og video-chat.

Plugin'en tillater GMail-brukere også å ringe til fasttelefon og mobiltelefon. Plugin'en er i tillegg integrert med Google sitt online sosiale nettverk *Orkut*, og Google sin personlige nettportal, *iGoogle*, slik at GMail-brukere også kan chatte med brukere av disse tjenestene.

Signalering og mediaoverføring håndteres av plugin'en. Alt GMail brukere trenger for å benytte tjenesten, er et kamera og en bredbåndsforbindelse. Dersom en av partene ikke har et kamera, så tillater tjenesten fremdeles en ènveis videosamtale eller en audio-sesjon. Også Google har vært opptatt å tillate flest mulig brukere å benytte VoIP-tjenesten, ettersom tjenesten er kryssplattformkompatibel og nettleseruavhengig [24].

Google Talk er en programvareapplikasjon tilbudt av Google. Google Talk tillater brukere å sende direktemeldinger, filoverføringer og å plassere anrop med VoIP. Denne applikasjonen er en desktop-applikasjon kun tilgjengelig for Windows-maskiner. Klienten er integrert med GMail, slik at applikasjonsbrukere også kan chatte eller utveksle innhold med GMail-brukere. Google Talk er i tillegg tilgjengelig som en mobil applikasjon for smarttelefoner. *Google Voice* er også tjeneste tilbudt av Google. Google Voice er en webbasert tjeneste som tillater brukere å plassere voice eller videosamtaler fra PC til PC. Tjenesten er forskjellig fra Google sin nettleser-plugin, men integrerer også med GMail. Google Voice er kun tilgjengelig i enkelte land [23].

Google tillater brukere å ringe med desktopklienten Google Talk, fra nettleseren med Google Voice, og også plassere voice og video samtaler med nettleser-plugin'en, video and voice plugin. I de to siste tilfellene tillater Google brukere å ringe uavhengig av en klient. Tjenestene er i tillegg rettet mot å tillate størst mulig brukermasse.

VoIP-tjenesten i VoIPBook som tillater brukere å ringe, kunne likeledes være en fullstendig integrert del av Facebook. Som vi har sett, så er en av forutsetningene for økt nettverkseffekt for en tjeneste at tjenesten kan benyttes av flest mulige brukere. VoIPBook-applikasjonen er både plattformuavhengig og nettleseruavhengig, men bruker en SIP-klient som et mindretall har installert. Vi har også sett at det ikke finnes en spesielt dominerende SIP-klient, ettersom at Skype har tatt over store deler av VoIP-markedet. Dersom vi hadde implementert VoIP-tjenester uavhengig av en SIP-klient, og som en integrert del av Facebook, så ville vi unngått denne begrensingen. Jeg vil tro at en slik integrasjon av VoIP-tjenester i Facebook hadde

vært vellykket, ettersom at etter Facebook introduserte tredjepartsutvikling har andre store selskaper som blant annet Google fulgt etter og åpnet for utvikling mot deres plattformer. Imidlertid så kreves det langt flere ressurser og mer tid å implementere en slik VoIP-stakk fra bunnen av og for hver nettleser og plattform, enn det som er mulig innenfor rammene av en slik oppgave.

Bibliografi

[1] Oracle and/or its affiliates. *Java Platform, Standard Edition 6 - API Specification*, 2010. http://download.oracle.com/javase/6/docs/api/.

[2] Michael Haenlein Andreas Kaplan. Users of the world, unite! the challenges and opportunities of social media. 2010. http://www.sciencedirect.com/science/.

[3] IT Security Blog Burim Bajraktari. Facebook "csrf"attack-full disclosure. 2009. http://bukibv.blogspot.com.

[4] Steve Campbell. How does facebook work? the nuts and bolts. Feb 2010. http://www.makeuseof.com/tag/facebook-work-nuts-bolts-technology-explained/.

[5] Nicholad Carlson. At last- the full story of how facebook was founded. *Business Insider*, 2010. http://www.businessinsider.com/.

[6] OS X Daily. Mac os x directory structure explained. Mars 2007. http://osxdaily.com/2007/03/30/mac-os-x-directory-structure-explained/.

[7] World Wide Web Consortium Dave Raggett. Client-side scripting and html. 1997. http://www.w3.org/TR/WD-script-970314.

[8] Skype Developers. *Skype*, Nov 2010. http://www.skype.com/intl/no/features/.

[9] Eric Eldon. Facebook employee count rising this year, up past 1300 already. Mars 2010. http://www.insidefacebook.com/.

[10] Glenn Engler. Facebook's growth. 2011. http://www.glennengler.com/fun-factoids/facebooks-growth/.

[11] Aditya Agarwal (Director of Engineering at Facebook). Scale at facebook. Mai 2010. http://www.infoq.com/presentations/Scale-at-Facebook.

[12] Github Facebook Development Team. *Facebook Andriod SDK*, 2010. http://github.com/facebook/facebook-android-sdk/.

[13] Github Facebook Development Team. *Facebook iOS SDK*, 2010. http://github.com/facebook/facebook-ios-sdk/.

[14] Github Facebook Development Team. *Facebook PHP SDK*, 2010. http://github.com/facebook/php-sdk/.

[15] Michael Fern. Is facebook worth 50b dollars? 2011. http://www.fernstrategy.com/.

[16] Internet Engineering Task Force. *RFC 768: UDP: User Datagram Protocol*, 1980. http://tools.ietf.org/html/rfc768.

[17] Internet Engineering Task Force. *RFC 768: TCP: Transmission Control Protocol*, 1981. http://tools.ietf.org/html/rfc793.

[18] Internet Engineering Task Force. *RFC 3261: Session Inition Protocol*, 2002. http://tools.ietf.org/html/rfc3261.

[19] Internet Engineering Task Force. *RFC 3550: RTP: A Transport Protocol for Real-Time Applications*, 2003. http://tools.ietf.org/html/rfc3550.

[20] Internet Engineering Task Force. *The Oauth 2.0 Protocol Framework*, Des 2010. http://tools.ietf.org/html/draft-ietf-oauth-v2-11.

[21] Network Working Group. *RFC 3824: Using E.164 numbers with the Session Initiation Protocol (SIP)*, 2004. http://www.packetizer.com/rfc/rfc3824/.

[22] Jonathan Rosenberg Henning Schulzrinne. The session initiation protocol: Providing advanced telephony services across the internet. www.cs.columbia.edu/.

[23] Google Inc. *Google Talk*, 2011. http://www.google.com/talk/about.html.

[24] Google Inc. Google voice and video chat. 2011. http://www.google.com/support/chat/.

[25] SJLabs Inc. *SJ Labs*, 1999. www.sjphone.org/.

[26] SJLabs Inc. *SJPhone User Guide*, 2007. www.sjlabs.com/sjp.html.

[27] Paul E. Jones. H.323 protocol overview. 2007. http://hive1.hive.packetizer.com/users/packetizer/papers/.

[28] Audun Jøsang. Cryptography. 2010. http://www.uio.no/.

[29] Audun Jøsang. Key management and pki. 2010. http://www.uio.no/.

[30] Niall Kennedy. Facebook's growing infrastruture spend. Mars 2009. http://www.niallkennedy.com/blog/.

[31] David Kirkpatrick. Help wanted: Adults on facebook. Rapport, 2008. cnn.money.com.

[32] Rich Miller. Facebook server count:60 000 or more. Juni 2010. http://www.datacenterknowledge.com/archives/2010/06/28/facebook-server-count-60000-or-more/.

[33] Rich Miller. Facebook to build it's own data center. Jan 2010. http://www.datacenterknowledge.com.

[34] MobileFish.com. Java quick guide, keytool. 2008. http://www.mobilefish.com/tutorials/java/.

[35] Microsoft Developer Network. Introduction to activex controls. 2011. http://msdn.microsoft.com/en-us/library/.

[36] Oracle. *Java(TM) Platform Standard Edition 6 API Specification*, 1993. http://download.oracle.com/javase/6/docs/api/.

[37] Oracle. Jarsigner - jar signing and verification tool. 2010. http://download.oracle.com/javase/.

[38] Oracle. Keytool - key and sertificate management tool. 2010. http://download.oracle.com/javase/.

[39] Sun Developer Network (SDN) Oracle. Applets. 2011. http://java.sun.com/applets/.

[40] Adam Ostrow. Facebook flyers 2.0: The promise and pitfalls of social networking ads. 2008. http://mashable.com/2007/10/22/facebook-flyers/.

[41] Adam Ostrow. Facebook announces the places-app to move into foursquares geolocation space. 2010. http://www.scmagazineuk.com/.

[42] Pingdom. Exploring the software behind facebook. Juni 2010. http://royal.pingdom.com/2010/06/18/the-software-behind-facebook/.

[43] Dan Raywook. Facebook announces the 'places' application to move into foursquare's geolocation space. 2010. http://www.scmagazineuk.com/.

[44] Ruchi Sanghvi. Facebook gets a facelift. Rapport, 2008. http://blogs.wsj.com/.

[45] Sun Developer Network (SDN). Applets. 2010. http://java.sun.com/applets/.

[46] Arun Sundararajan. Network effects. 2006. http://oz.stern.nyu.edu/io/network.html.

[47] Apple Development Team. Mac os x reference library - application bundles. 2010. http://developer.apple.com/library/mac.

[48] Facebook Developemnt Team. *Old Rest API*, 2010. http://developers.facebook.com/docs/reference/rest/.

[49] Facebook Developemnt Team. *Social Plugins*, 2010. http://developers.facebook.com/plugins.

[50] Facebook Developer Team. Facebook developer wiki. 2010. http://developers.facebook.com/.

[51] Facebook Developer Team. Facebook press room. 2010. http://www.facebook.com/press.php.

[52] Facebook Developer Team. Facebook tos. Rapport, 2010. http://www.facebook.com/terms.php?ref=pf.

[53] Facebook Development Team. *Authentication*, 2010. http://developers.facebook.com/docs/authentication/.

[54] Facebook Development Team. *Documentation*, 2010. http://developers.facebook.com/docs/.

[55] Facebook Development Team. *Facebook C Sharp SDK*, 2010. http://github.com/facebook/csharp-sdk/.

[56] Facebook Development Team. *Facebook for Websites*, 2010. http://developers.facebook.com/docs/.

[57] Facebook Development Team. *Facebook Query Language*, 2010. http://developers.facebook.com/docs/reference/fql/.

[58] Facebook Development Team. *Graph API*, 2010. http://developers.facebook.com/docs/api.

[59] Facebook Development Team. *Integrating with Facebook Chat*, 2010. http://developers.facebook.com/docs/chat.

[60] Facebook Development Team. *Javascript SDK*, 2010. http://developers.facebook.com/docs/reference/javascript/.

[61] Facebook Development Team. *Javascript SDK, XFBML methods*, 2010. http://developers.facebook.com/docs/reference/javascript/FB.XFBML.parse.

[62] Facebook Development Team. *Open Graph Protocol*, 2010. http://developers.facebook.com/docs/opengraph.

[63] Facebook Development Team. *Open Source*, 2010. http://developers.facebook.com/opensource/.

[64] Facebook Development Team. Privacy policy. Rapport, 2010.

[65] Pål Unanue-Zahl. 1,5 millioner nordmenn på facebook. Rapport, 2009. www.vg.no.

[66] International Telecommunication Union. *E.164*, 2009. http://www.itu.int/.

[67] W3Schools.com. Html applet tag. 2011. http://www.w3schools.com/tags/tag_applet.asp.

[68] Wikipedia. Activex. 2010. http://en.wikipedia.org/wiki/ActiveX.

[69] Wikipedia. Criticism of facebook. 2010. http://en.wikipedia.org/wiki/Criticism_of_Facebook.

[70] Wikipedia. Facebook. 2010. http://en.wikipedia.org/wiki/Facebook.

[71] Wikipedia. Facebook credits. 2010. http://en.wikipedia.org/wiki/Facebook_Credits.

[72] Wikipedia. Filesystem hierarchy standard. 2010. http://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard.

[73] Wikipedia. Java applet. 2010.

[74] Wikipedia. Network effect. 2010. http://en.wikipedia.org/wiki/Network_effect.

[75] Wikipedia. Session inition protocol. 2010. http://en.wikipedia.org/wiki/Session_Initiation_Protocol.

[76] Wikipedia. Social networking service. 2010. http://en.wikipedia.org/wiki/Online_social_networks.

[77] Wikipedia. Voice over internet protocol. Mars 2010.

[78] Wikipedia. Applets. 2011. http://en.wikipedia.org/wiki/Applet.

[79] Brian Womack. Facebook sees fourfold jump in number of advertisers since 2009. 2010. www.businessweek.com.

Kapittel 12

Vedlegg A

Programmeringskode for VoIPBook-applikasjonen og AppLauncher-appleten.

Filen index.php:

```
<?php

require 'facebook.php';

$appId = '110822422284818';
$appapikey = '07f588d1525c12cbd243487cf127fc3a';
$secret = 'c9e2426a4ca370a3eb44f0f6ba55f14f';
$base_domain = 'http://sonamr.at.ifi.uio.no/php/Master/';
$app_domain = 'http://apps.facebook.com/iptelephonytester/';
$app_name = 'The VoIP Integration';
$app_ext = 'iptelephonytester';
$me = 'http://graph.facebook.com/me';
$inserted = $_POST['insert'];
$sipURI =  $_POST['register'];
$invited = $_POST['ids'];

$facebook = new Facebook(array(
  'appId'  => '110822422284818',
  'secret' => 'c9e2426a4ca370a3eb44f0f6ba55f14f',
  'cookie' => true,
));

$facebook->setAppId($appId);
$facebook->setApiSecret($secret);
$facebook->setBaseDomain($base_domain);
$token = $facebook->getAccessToken();

try {
  $me = $facebook->api('/me');
} catch (FacebookApiException $e) {
  error_log($e);
}

$logoutURL= $facebook->getLogoutUrl(array('next' => $app_domain));
$loginURL = $facebook->getLoginUrl(array(
'iframe' => 1,
'fbconnect' => 0,
'req_perms' => 'manage_pages, publish_stream, sms, user_about_me,
email, user_online_presence, offline_access',
'next' => $app_domain,
'cancel_url' => $base_domain ));

if ($facebook->getSession()) {
  echo '<a href="' . $facebook->getLogoutUrl(array('next' => $app_domain)) . '">Logout</a>';
} else {
  echo '<a href="' . $facebook->getLoginUrl(array
('iframe' => 1,
 'fbconnect' => 0,
```

```
 'req_perms' => 'manage_pages, read_insights, publish_stream,
user_about_me, email, user_online_presence, offline_access',
 'next' => $app_domain,
 'cancel_url' => $base_domain )) . '">Login</a>';
}

$session = $facebook->getSession();

//<!OPPSETT AV LINKER TIL ANDRE SIDER I APPLIKASJONEN>
//<!------------------------------------------------------------------------------------------>

$my_info = $facebook->api
('/me?fields=id,name,gender,birthday,email,location,about,link,work,education');

$user_id = $my_info['id'];
$user_name = $my_info['name'];
$gender = $my_info['gender'];
$birthday = $my_info['birthday'];
$email=$my_info['email'];
$hometown=$my_info['location'];
$home=$my_info['name'];
$abt=$my_info['about'];
$link=$my_info['link'];
$wor=$my_info['work'];
$work=$wor['0'];
$employerid=$work['employer'];
$employer=$employerid['name'];
$pos=$work['position'];
$position=$pos['name'];
$ed=$my_info['education'];
$education=$ed['0'];
$school=$education['school'];
$schoolname=$school['name'];
$area=$education['concentration'];
$con=$area['0'];
$field=$con['name'];

echo "<a href='http://sonamr.at.ifi.uio.no/php/Master/me.php?id=$user_id&name=$user_name
&gender=$gender&birthday=$birthday&email=$email&website=$website&hometown=$home&abt=$abt
&link=$link&employer=$employer&position=$position
&school=$schoolname&field=$field'> Persona  Information </a>";

$uid = $session['uid'];
$name=$me['name'];

echo "<a href='http://sonamr.at.ifi.uio.no/php/Master/friends.php?
fbid=$uid&person=$name'> My Contacts </a>";

$message = "I am calling all my friends with VoIP through VoIPBook. You should join!";
$invite = "https://api.facebook.com/method/stream.publish?
message=" . $message . "&uid=" . $uid . "&access_token=" . $token;

echo '<a href="' . $invite . '"> Invite your Friends  </a>';

echo "<a href='http://sonamr.at.ifi.uio.no/php/Master/applet.php?
fbid=$uid&person=$name'> Place a Call </a>";

//<!OPPSETT AV APPLIKASJOEN OG STYLESHEET>
//<!------------------------------------------------------------------------------------------>
?>
<!doctype html>
<html xmlns:fb="http://www.facebook.com/2008/fbml">
<head>
<title>VoIPBook</title>
<link rel="stylesheet" type="text/css" href="stylesheet.css" />
</head>
<body>

<div id="fb-root"></div>
<script>

window.fbAsyncInit = function() {
FB.init({
appId : '<?php echo $facebook->getAppId(); ?>',
session : <?php echo json_encode($session); ?>,
status : true, // check login status
cookie : true, // enable cookies to allow the server to access the session
xfbml : true // parse XFBML
});

// whenever the user logs in, we refresh the page
FB.Event.subscribe('auth.login', function() {
```

```
window.location.reload();
});
};

(function() {
var e = document.createElement('script');
e.src = document.location.protocol + '//connect.facebook.net/en_US/all.js';
e.async = true;
document.getElementById('fb-root').appendChild(e);
}());

</script>

<h1><a href="home.php" class="fbody">VoIPBook</a></h1> <!hvis man trykker - leder hjem>

<!SKRIVER UT OM MEG>
//<!-------------------------------------------------------------------------------------------->

<?php if ($me): ?>

<h3>Your Profile</h3>
<img src="https://graph.facebook.com/<?php echo $uid; ?>/picture?type=large">
<pre><?php echo 'Name: '; print_r($me['name']);?></pre>
<pre><?php echo 'FBID: '; print_r($me['id']); ?></pre>
<pre><?php echo 'DoB: '; print_r($me['birthday']); ?></pre>
<pre><?php echo 'Email: '; print_r($me['email']); ?></pre>

<?php else: ?>
<strong><em>You are not Connected.</em></strong>
<?php endif ?>
<pre><?php

$existingFile = "FacebookAdressFile" ."-" . $me['id'] . ".txt";
$existingHandle = fopen($existingFile, 'r');

if ($existingHandle != FALSE) {
  $contents = fread($existingHandle, filesize($existingFile));
  echo 'Your current contact adresses: ' . "\n";
  echo $contents;
  fclose($existingHandle);

 }

 ?></pre>

<!HER HENTES VENNER UT OG TELLES OPP - KAN EVT SKRIVES UT >
//<!-------------------------------------------------------------------------------------------->

<pre><?php $friends = $facebook->api('/me/friends?fields=id,name,email');?></pre>
<pre><?php $data = $friends['data'];?></pre> <!stripper unødig info>
<pre><?php

$myFile = "FacebookFriendFile" ."-" . $me['id'] . ".txt";
$fileHandle = fopen($myFile, 'w') or die("can't open file");

$friendCounter = 0;
for ($i=0; $i<= 1000; $i++) {
  $f = $data[$i]; //henter ut arrayen på plass i
  if ($f == null) break; //sjekker om ikke fler da termineres løkken
  $friendCounter++; //øker teller
  //print_r($f['id']);
  //echo ' = ';
  //print_r($f['name']);
  $data_string = $f['id'] . " : " . $f['name'] . ";";
  fwrite($fileHandle, $data_string);
  fwrite($fileHandle, "\n");

  ?></pre>
  <pre><?
   }//end for
fclose($fileHandle);
echo 'Total friends: ' . $friendCounter;

?><pre>

<! LINKEN NEDENFOR GIR BRUKEREN ANLEDNING TIL Å LASTE NED EN FIL MED VENNERS INFROMASJON >
//<!-------------------------------------------------------------------------------------------->

<a href="http://sonamr.at.ifi.uio.no/php/Master/<?=$myFile?>">
Download Information About Your Friends</a>

  <h3>Add a contact adress: </h3>
```

```
<!LAGER TEKSTOMRÅDE FOR Å LEGGE TIL SIP-ADRESSER>
//<!------------------------------------------------------------------------------------------>
<pre><?php if (!isset($_POST['submit'])) { ?><pre>

<div id="Insert URI">
<form name="InsertURI" action="<?php echo $PHP_SELF;?>" method="post">
<textarea rows="4" cols="50" name="insert" wrap="physical"> Enter your SIP-URI </textarea>
<br/>
  <input type="submit" value="Submit" name="submit" />

  <pre><?php } else {
  echo ' You have successfully inserted following contact-adress: ';
  echo $inserted;
  $myAdrFile = "FacebookAdressFile" ."-" . $me['id'] . ".txt";
  $existingHandle = fopen($myAdrFile, 'a');
  if ($existingHandle != FALSE) {
    fwrite($existingHandle, $inserted . "\n");
    fclose($existingHandle);
  }
  else {
    $newHandle = fopen($myAdrFile, 'w') or die("Can't open file");
    fwrite($newHandle, $inserted . "\n");
    fclose($newHandle);
  }

 } ?><pre>

<br/>
</form>
</div>
</body>
</html>
```

Filen friends.php:

```
<pre><?php

$id = $_GET['fbid'];
$name = $_GET['person'];
echo '<img src="https://graph.facebook.com/' .$id . '/picture"/>';
echo $name . "(" . $id . ")\n\r";

$myAdrFile = "FacebookAdressFile" ."-" . $id . ".txt";
$myHandle = fopen($myAdrFile, 'r');

if ($myHandle != FALSE) {
  $contents = fread($myHandle, filesize($myAdrFile));
  echo 'YOUR CURRENT CONTACT-ADRESSES: ' . "\n";
  echo $contents;
  fclose($myHandle);

 }

$existingFile =  "FacebookFriendFile" ."-" . $id . ".txt";
$existingHandle = fopen($existingFile, 'r');

if ($existingHandle != FALSE) {
  echo "\n" . 'YOUR CURRENT CONTACTS: ' . "\n";
  $contents = fread($existingHandle, filesize($existingFile));
  $words = explode(";", $contents);
  foreach ($words as $word) {
    echo ($word);
    $tmp = explode(":", $word);
    $fbid = trim($tmp[0]);
    findAdr($fbid);

  }
   fclose($existingHandle);
 }

else echo "Cannot open file.";

function findAdr($fbid) {
```

```
  if (!empty($fbid))  echo '<img src="https://graph.facebook.com/' .$fbid . '/picture"/>';

  $frFile = "FacebookAdressFile" ."-" . $fbid . ".txt";
  $frHandle = fopen($frFile, 'r');

  if ($frHandle != FALSE) {
    $contents = fread($frHandle, filesize($frFile));
    echo " contact: ";
    $adrs = explode("\n", $contents);
    foreach ($adrs as $adr) echo $adr . " ";
  }
  fclose($frHandle);
}

?></pre>
```

Filen, applet.php:

```
<pre><?php

$id = $_GET['fbid'];
$name = $_GET['person'];
$counter = 0;
$num_fr = 0;
$friends = array();
$friends_adrs = array();
print '<img src="https://graph.facebook.com/' .$id . '/picture"/>';
echo $name;

$existingFile =  "FacebookFriendFile" ."-" . $id . ".txt";
$existingHandle = fopen($existingFile, 'r');

if ($existingHandle != FALSE) {

  $contents = fread($existingHandle, filesize($existingFile));
  $words = explode(";", $contents);
  foreach ($words as $word) {
    $tmp = explode(":", $word);
    $fbid = trim($tmp[0]);
    findAdr($fbid, $tmp[1]);

  }

  fclose($existingHandle);
 }

else echo "Cannot open file.";

function findAdr($fbid, $name) {

  global $counter;
  global $friends_adrs;
  global $num_fr;
  global $friends;

  $frFile = "FacebookAdressFile" ."-" . $fbid . ".txt";
  $frHandle = fopen($frFile, 'r');

  if ($frHandle != FALSE) {
    $contents = fread($frHandle, filesize($frFile));
    $adrs = explode("\n", $contents);
    foreach ($adrs as $adr)  {
      if (!empty($adr)) {
$friends[$num_fr++] = $name . " "; å
$friends_adrs[$counter++] = $adr;
      }
    }
  }
   fclose($frHandle);
}

?></pre>
```

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
<TITLE>Java Applet</TITLE>
</HEAD>
<BODY>
<H1>Place a call with VoIPBook</H1>
<P><APPLET code="AppLauncher.class" archive="SignedAppLauncher.jar" WIDTH="600" HEIGHT="400">
<param name="numberOfFriends" value="<?php echo $counter ?>"/>
  <?php for ($i=0; $i<$counter; $i++) {?>
<param name="display<?php echo $i; ?>" value="Place a call to
<?php echo $friends[$i]; echo '( ' . $friends_adrs[$i] . ' )'; ?>"/>
  <?php } ?>
<param name="display" value="Place a call to <?php echo $name; ?>"/>
<param name="location" value="cmd.exe"/>
</APPLET></P>
</script>
</BODY>
</HTML>
```

Filen, AppLauncher.java

```
import java.applet.Applet;
import java.util.ArrayList;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.lang.*;
import java.net.URL;
import java.io.*;

public class AppLauncher extends Applet implements ActionListener {


    ArrayList<String> friends;
    ArrayList<JButton> buttons;
    ArrayList<Button> mac_buttons;

    boolean appErr = false;

    public static void main (String[] args) {
String s = System.getProperty("os.name");
System.out.println(s);
String t = System.getenv("HOME");
System.out.println(t);
    }

//<!-------------------------------------------------------------------------------------------->
    public void init() {

friends = new ArrayList<String>();
buttons = new ArrayList<JButton>();

setLayout(new FlowLayout());
setForeground(Color.BLACK);

    }
//<!-------------------------------------------------------------------------------------------->
    public void start() {

int numFriends = Integer.parseInt(this.getParameter("numberOfFriends"));
System.out.println(numFriends);
ImageIcon icon = new ImageIcon("pr_image.gif");
String os = System.getProperty("os.name");

for (int i=0;i<numFriends;i++)
    {
String friend = this.getParameter("display"+i);
friends.add(friend);

try {
    //URL image = new URL(url);
```

```
    //ImageIcon im = new ImageIcon(image);
    if (os.startsWith("Mac OS X")) {

Button b = new Button(friend);
b.setPreferredSize(new Dimension(500, 75));
b.setActionCommand(friend);
b.addActionListener(this);
mac_buttons.add(b);

    }

    else {

    JButton jb = new JButton(friend, icon);
    jb.setPreferredSize(new Dimension(500, 75));
    jb.setActionCommand(friend);
    jb.addActionListener(this);
    buttons.add(jb);

    }
}
catch (Exception e) {
    ;
}

    }

if (os.startsWith("Mac OS X")) {
    for (int i=0;i<mac_buttons.size();i++) {
  add((Button)mac_buttons.get(i));
    }
}
else {
    for (int i=0;i<buttons.size();i++) {
    add((JButton)buttons.get(i));
          }
}

    }
//<!------------------------------------------------------------------------------------------>
    public void actionPerformed(ActionEvent evt) {

String label = evt.getActionCommand();
String callee = parse(label);
launch(callee);
    }
//<!------------------------------------------------------------------------------------------>
    public void launch (String callee) {

String osType = System.getProperty("os.name");

if (osType.startsWith("Windows")) {
    windows(callee);
        }
else if (osType.startsWith("Linux")) {
    linux(callee);
}
else if (osType.startsWith("Mac OS X")) {
    macOSX(callee);
}
else {
    appErr = true;
    repaint();
}
    }
//<!------------------------------------------------------------------------------------------>
    public void windows (String callee) {

String programfiles = System.getenv("PROGRAMFILES");
programfiles += "\\SJphone 1.65";
//String[] cmd = {"SJphone.exe", ""+callee};
String [] cmd = {"cmd", "/C", "start", "cd programfiles", "SJPhone 1.65.lnk ", ""+callee};

try {
    //Runtime.getRuntime().exec(cmd, null, new File(programfiles));
    Runtime.getRuntime().exec(cmd);
}
catch (Exception e) {
    appErr = true;
    System.out.println
            ("\nProblem trying to launch application SJPhone, and calling "+callee+".\n");;
    e.printStackTrace();
```

```
    repaint();
}
    }
//<!--------------------------------------------------------------------------------------------->
    public void linux (String callee) {

System.out.println("OS is Linux.");
String[] command = {"SJphoneLnx-299a/sjphone", ""+callee};
String home = System.getenv("HOME");

try {
    System.out.println("OS is Linux.");
    Runtime.getRuntime().exec(command, null, new File(home));
        }                               }
catch (Exception e) {
    appErr = true;
    System.out.println
            ("\nProblem trying to launch application SJPhone, and calling "+callee+".\n");;
    e.printStackTrace();
    repaint();
        }
    }
//<!--------------------------------------------------------------------------------------------->
    public void macOSX (String callee) {

System.out.println("OS is Mac.");
String [] cmd = {"open", "SJphone.app", ""+callee};
String home = System.getenv("HOME");
home += "/Applications";
System.out.println(home);

try {
    Runtime.getRuntime().exec(cmd, null, new File(home));
}

catch (Exception e) {
    appErr = true;
    System.out.println
           ("\nProblem trying to launch application SJPhone, and calling "+callee+".\n");;
    e.printStackTrace();
    repaint();
}

    }
//<!--------------------------------------------------------------------------------------------->
    public void paint(Graphics g) {

if (appErr) {
    g.drawString("Could not place call. Make sure SJPhone is installed correctly.
            This applet can only be run on Windows, Linux or Mac OS X platforms.",100,100);
}
    }
//<!--------------------------------------------------------------------------------------------->
    public String parse (String s) {

String tmp = "";

for (int i=0; i<s.length(); i++) {
    if (s.charAt(i) == '(') {
while (s.charAt(++i) != ')') {
    tmp += s.charAt(i);
}
    }
}
String trimmed = tmp.trim();
String fixed = "sip:"+trimmed;
return fixed;
    }
}
```